I giochi del computer

di Corrado Giustozzi

Nelle scorse edizioni di MCgiochi abbiamo discusso, più o meno approfonditamente, dei vari tipi di giochi disponibili sul computer: dagli arcade agli adventure, passando anche per i boardgame. Oggi vogliamo invece parlare di una categoria di giochi completamente diversa dalle altre già viste. La differenza è semplice ma fondamentale: in questi giochi il giocatore è il computer, e l'uomo... sta a guardare.

Sono giochi in senso lato, nei quali il divertimento non consiste nell'interagire con il computer ma esattamente l'opposto: nell'osservare, senza intervenire, ciò che il computer fa. Stiamo ovviamente parlando di tutta quella classe di programmi ludici che va sotto il termine di simulazioni, generalmente discendenti da serissimi programmi di ricerca adattati poi a pura ricreazione. In questi programmi, al contrario degli arcade o degli adventure, l'uomo non compie alcun ruolo attivo, non influenza il corso dell'elaborazione: tutt'al più stabilisce dei parametri iniziali, delle condizioni al contorno, da cui si evolve, senza più alcun intervento esterno, la simulazione. Può sembrare strano ma programmi di questo tipo sono piuttosto divertenti e consentono anche diversi tipi di "gioco", molto diversi l'uno dall'altro. Crediamo che l'argomento non sia mai stato trattato con il rispetto e la completezza dovuti, per cui abbiamo deciso di mettere in cantiere una serie di puntate dedicate a questi programmi. In questa puntata introduttiva discuteremo le caratteristiche comuni della famiglia e faremo la conoscenza con i membri più interessanti. Successivamente dedicheremo intere puntate ai giochi più rappresentativi, imparando a conoscerli più da vicino. Ci auguriamo fin d'ora di riscuotere il vostro interesse, e speriamo che ci seguiate anche attivamente, inviandoci cioè interventi, idee e, perché no, programmi.

La classe dei programmi di simulazione matematica nasce, come accennato, nei laboratori di ricerca e nelle università fin dai primordi dell'era informatica. Scienziati e tecnici si accorgono subito che il calcolatore è un potentissimo mezzo per indagare la realtà, alternativo agli esperimenti "tradizionali"; col calcolatore si può "simulare" una realtà fisica, eseguendo esperimenti validi altrettanto quanto quelli di laboratorio. Come? Semplice. Se di un certo fenomeno fisico si conosce la teoria matematica, basta impostare le equazioni in un programma ad hoc per poter eseguire quanti "esperimenti" si vuole, con le più disparate condizioni al contorno. Se non si conosce una teoria rigorosa, il calcolatore va ugualmente bene: serve a verificare la consistenza delle ipotesi proposte, confrontando le simulazioni con i risultati reali.

C'è di più: il calcolatore permette di simulare anche condizioni ideali e/o esperimenti concettuali, cose che in nessun laboratorio si possono fare. Condizioni di attrito nullo, di zero assoluto, comportamento di monopoli magnetici, di superfici in spazi non euclidei o di ordini superiori... tutto ciò che può essere descritto matematicamente va bene come oggetto di una simulazione. Anche entità non rigorose come sistemi biologici o sociali vengono simulati, a partire dai semplici modelli preda-predatore fino ai complessissimi modelli econometrici del mondo intero sviluppati al MIT per conto del Club di Roma. Il trucco è conoscere (o credere di conoscere, o saper approssimare) le relazioni matematiche che quantificano le interazioni delle varie parti componenti il sistema in esame, e quelle del sistema con l'esterno: ciò che si chiama il modello matematico del fenomeno in esame. Tutto può essere rappresentato con un modello matematico: dalla struttura di un grattacielo alla crescita di un fiocco di neve, dal comportamento dei Lemming al movimento di una sfera in moto in un fluido viscoso, dal sistema economico dell'impero egiziano all'orbita di un neutrone attorno ad un buco nero. Simulare al calcolatore significa scrivere un programma che implementi le nostre conoscenze del modello matematico di un dato sistema, fornire al programma dati di partenza ragionevoli o, almeno, consistenti e... stare a vedere che succede. Vedere cosa succede al nostro grattacielo se c'è un vento di 400 Km/h, al nostro fiocco di neve se non c'è gravità o ai nostri Lemming se ci sono pochi licheni e troppi lupi; seguire l'evoluzione nel tempo, o lungo un'altra grandezza, del nostro sistema. Si scorge subito che molti aspetti della simulazione al calcolatore hanno un carattere ludico neanche tanto riposto: spesso seguire l'evoluzione di un sistema simulato è piuttosto divertente, oltre che interessante.

Uno dei primi giochi esplicitamente tali basato su una semplice simulazione al calcolatore è l'ormai famosissimo Life (Vita), inventato dal matematico John Horton Conway e diffuso in tutto il mondo dalla fantastica penna di Martin Gardner. Basato su lavori precedenti ispirati ad una serissima teoria detta degli automi cellulari, dovuta al celebre matematico Von Neumann (papà di tutti gli informatici), Life simula la crescita di organismi uni-

cellulari per mezzo di semplici regole di nascita e di morte. È sorprendente vedere come a partire da un insieme di entità (i punti che rappresentano le cellule) e di informazioni (le regole) apparentemente poco più che banali, si possano sviluppare delle configurazioni dalla incredibile complessità; Life è ormai uno degli esempi classici che i testi di teoria dei sistemi portano per dimostrare come la complessità di un sistema sia alle volte spaventosamente alta anche quando le "leggi" che regolano il sistema sono molto semplici.

Un altro gioco della stessa famiglia di Life è Worms (Vermi), che simula il percorso di un verme che procede scavando una lunga galleria nel terreno; nato anch'esso come sottoprodotto di lavori serissimi (nientedimeno che dagli studi di Seymour Pappert sul Logo al MIT), è piuttosto sorprendente per la complessità dei percorsi tracciati dal verme e per la repentina subitaneità con cui questo, improvvisamente muore.

I giochi del genere di Life e Worms sono ormai diversi, e sono fra i più anziani della famiglia che stiamo imparando a conoscere; una classe molto più recente, e parecchio differente, ha come rappresentante un gioco denominato Core Wars (Guerra del Nucleo), ideato da A.K. Dewdney, un professore di scienza dei calcolatori in un'università canadese. In esso si ipotizza una situazione alla "Tron" in cui due programmi si danno battaglia nella memoria centrale di un calcolatore ("core", appunto) cercando di danneggiarsi a vicenda. Ciò può essere paragonato allo scontro di due organismi (due virus, due robot?) ognuno dei quali tende ad ottenere la morte (o invasione) dell'altro ed il completo dominio sull'ambiente. Il gioco, presentato in Italia per la prima volta sulla rivista "Le Scienze" di luglio '84, è piuttosto interessante; tanto che (notizia in anteprima...) abbiamo deciso di realizzarne una versione che verrà presto presentata su queste pagine. Anche in Core Wars, come nei giochi visti in precedenza, l'intervento umano è limitato al minimo: una volta preparati i due programmi (secondo le strategie più personali), si mettono nel campo di battaglia e lì si lasciano a combattere finché uno dei due non soccombe; durante la lotta gli umani sono solo spettatori, e non possono influenzare in alcun modo il comportamento delle proprie creature.

Un ultimo aspetto della simulazione ludica, piuttosto lontano da quanto abbiamo visto finora ma non per questo meno interessante, è quello consistente nell'usare il computer per la ricerca di soluzioni di particolari problemi o rompicapi più o meno matematici: dal problema delle otto regine, magari generalizzato in vari modi, a quello del percorso di cavallo, dalla ricerca dei numeri con particolari proprietà (perfetti, primi, amicabili), a quello di speciali tassellature nel piano. Ma qui il discorso si fa troppo vasto e rischia di portarci troppo lontano, per cui è meglio fermarsi.

Bene, nel corso di questo primo incontro con i giochi di simulazione matematica abbiamo incontrato diversi rappresentanti di una vasta classe di programmi divertenti, imparando a conoscere a grosse linee le caratteristiche comuni di essi. Nelle prossime puntate ci occuperemo più in dettaglio dei singoli giochi e delle loro varianti, vedendo come implementarli e quali risultati si possono ottenere. Se ritenete di avere suggerimenti interessanti da offrirci in merito non esitate a scriverci. Nel frattempo, buon divertimento! MC

Sotto l'albero di Natale

Dall'Inghilterra è in arrivo la solita ondata di software natalizio; vediamo alcune fra le novità principali.

La prima riguarda la conversione di un grande successo dello Spectrum per il Commodore: anche i possessori di C 64 potranno finalmente divertirsi con Ant Attack, il celebre programma della Quicksilva.

Sempre di Sandy White, l'autore di Ant Attack, è pronto Zombie Zombie, questa volta per lo Spectrum 48K.

La Melbourne House ha sparato molte delle sue cartucce in settembre, ma ha ancora qualcosina da mostrare: gli amanti dei film di Dracula AND i possessori di un 64 potranno rabbrividire con Castle Of Terror, una grande avventura grafica che vede protagonista proprio il famoso Conte succhiasangue.

L'Automata presenterà invece il primo gioco con commento musicale esterno: all'acquirente verrà infatti data una cassetta con incisa la colonna sonora originale di Deus ex Machina (questo è il titolo); il programma stesso segnalerà il momento di far partire il nastro, in modo da riuscire a sincronizzare l'azione con la musica.

Due nuovi titoli anche dalla Ultimate: Underwurdle e Knight Lore, entrambi per lo Spectrum.

Scott Adams ha terminato la sua nuova avventura, questa volta sarà l'uomo ragno a darsi da fare, mentre pare che siano pronti ad entrare in azione anche i fantastici 4.

Fra tanti nomi vecchi ecco infine un nome nuovo (per il mercato inglese, beninteso): in contemporanea al lancio del computer MSX la Konami presenterà buona parte dalla sua vasta gamma di giochi per questo standard.

Polsiere da sala giochi

Si sa che gli americani sono imbattibili per inventare gli accessori ed i gadget più folli, ma questa volta devono aver stabilito proprio un record.

Immaginate la scena: siete impegnatissimi con l'ultimo gioco arrivato nella vostra arcade preferita, l'azione è frenetica, il sudore vi cola lungo le braccia, lungo le mani, cortocircuita il joystick e addio record.

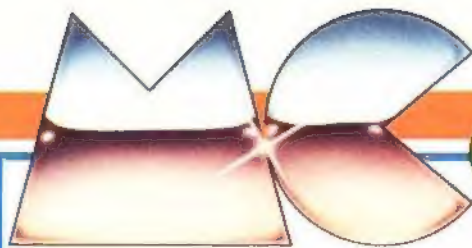
Solo a pensarci viene da meditare il suicidio, vero? Ma niente paura, perché la Sweet Gum di Miami ha messo in commercio i Computer Jock, sorta di polsiere da tennis adattissime per impedire questo tipo di infortuni.

Giusto per rendersi ancora più utili i Computer Jocks incorporano un orologio digitale ed una tasca, munita di cerniera, dove tenere i soldi per le partite.

Parola, non è uno scherzo! Quando vi sarete finalmente resi conto dell'importanza di questo fondamentale accessorio potrete ordinarne uno per una dozzina di dollari alla Sweet Gum, 15490. N.W. 7th Ave., Miami.

Oxford: progettare giochi 3D per il 64

A parte i vari compilatori per i computer Commodore (per il Basic, con Petspeed, e per il Pascal, in una versione grafica), la Oxford fa anche giochi. Il più importante dell'ultima generazione è Turbo, che si svolge in una pista automobilistica a tre dimensioni, sfruttando con il massimo realismo le possibilità grafiche e sonore del C64.

Ma non basta: nella prossima produzione è stato annunciato un package — derivato dalla lavorazione di Turbo — che permetterà a tutti di disegnare la propria pista, eventualmente per inserirla nei giochi di propria realizzazione. A completare la nuova serie di tool verranno commercializzati anche add-on grafici ed editori di schermo.

Life

di Corrado Giustozzi

Dopo l'introduzione del mese scorso incominciamo da questo mese ad addentrarci un pochino nel mondo di quei giochi in cui il computer svolge il ruolo principale: le simulazioni. Inizieremo con quello che è forse il più famoso gioco matematico di simulazione: Life (Vita), di cui abbiamo brevemente accennato la storia nell'articolo precedente.

Inventato dal matematico John Horton Conway verso la fine degli anni '60, ma derivato da studi teorici di Von Neumann e Ulam risalenti agli anni '50, Life è un "gioco" in senso piuttosto lato: non vi sono concorrenti e nessuno vince o perde; si tratta, se volete, di un divertimento concettuale, come più o meno tutti i giochi di simulazione di cui parleremo nel seguito. Scopo del gioco è solo seguire l'evoluzione nel tempo e nello spazio di alcune configurazioni di punti soggette a determinate regole. La struttura di base del gioco, il punto, è astratta, ma viene generalmente concretizzata immaginando che i punti rappresentino cellule ed organismi consimili. Il "mondo" in cui si svolge Life è...piuttosto piatto, nel vero senso della parola: tutto si svolge su di un piano, per di più illimitato in tutte le direzioni. Questo piano infinito è in realtà un reticolo, ossia è suddiviso in celle quadrate; a seconda del gusto personale si può pensare che le cellule giacciano sulle intersezioni del reticolo o che siano le celle stesse a costituire gli organismi unicellulari del gioco. Nella seconda versione (che noi adotteremo) le cellule sono quindi come le caselle di una scacchiera infinita. In ogni caso la cosa importante è che lo spazio in cui si svolge Life non è continuo, ma discreto: ogni cellula ha una posizione rigorosamente definita nel reticolo, e non può spostarsi nel resto del piano.

In questo strano mondo bidimensionale le cellule nascono, vivono e muoiono, creando aggregati che mutano nel tempo a seconda di quanti e quali individui sono vivi in un dato momento. Vita e morte delle cellule sono gli eventi base del gioco, e vanno quindi visti più da vicino. In Life una cellula può assumere solo due stati, che generalmente si indicano con vita e morte: in altre parole una cellula in un dato momento può essere solo viva o morta, senza alternative e senza ambiguità. Quando è viva la si rappresenta con una casella colorata, quanto è morta con una casella vuota. Naturalmente sono possibili dei cambiamenti di stato, ossia una cellula morta può diventare viva (si dice che nasce) e, viceversa, una cellula viva può morire. Esiste un insieme ben preciso di regole che ci mettono in grado di decidere la nascita o la morte di ogni cellula: sono le istruzioni del gioco, che vedremo in dettaglio fra poco. Come lo spazio, anche il tempo in Life è discreto: non procede con continuità ma a balzi, ad intervalli ben precisi. Fra l'uno e l'altro di questi intervalli avvengono tutti i cambiamenti nella popolazione: tutte le cellule che, in base alle regole, devono nascere nascono, tutte quelle che devono morire muoiono. Tutti gli eventi avvengono istantaneamente e contemporaneamente. Il risultato è una nuova "generazione" di cellule, che di solito è piuttosto diversa dalla precedente in quanto a numero di individui e tipo di aggregazione; si può quindi tranquillamente parlare di "evoluzione" della colonia originaria.

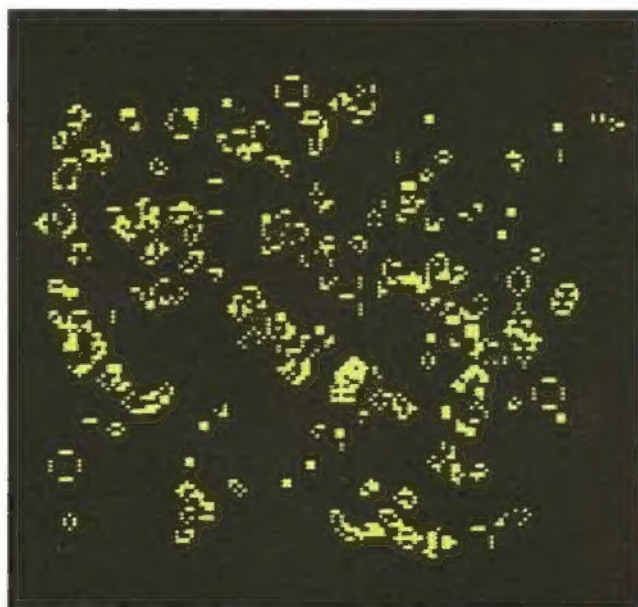
Passiamo quindi a parlare delle fatidiche leggi che regolano la vita e la morte degli organismi unicellulari di Life. Ci si potrebbe aspettare un insieme piuttosto complesso di condizioni, ma non è così: lo stato di una cellula è influenzato semplicemente dal suo e da quello dei suoi vicini, nel modo che vediamo subito. Innanzitutto è chiaro dalla disposizione a scacchiera delle cellule che ognuna di esse ha esattamente otto "vicini", ossia otto altre cellule adiacenti: due in verticale, due in orizzontale e quattro ai vertici. Per stabilire il futuro di una cellula basta contare quanti dei suoi vicini sono vivi, e vedere in che stato è la cellula attualmente. Se è viva ed ha due o tre vicini vivi rimane viva, altrimenti muore. Se è morta ed ha esattamente tre vicini vivi nasce, in caso contrario rimane morta. (La terminologia che si usa in caso di morte di una cellula è piuttosto espressiva: nel caso abbia meno di due vicini vivi si dice che muore per isolamento, se ne ha più di tre muore invece per sovrappopolazione). Le regole sembrano banali, ma il comportamento della popolazione che le segue non lo è affatto. Anzi, è incredibile constatare come un insieme talmente semplice di regole possa creare configurazioni dalla complessità enorme e dagli imprevedibili sviluppi; il punto affascinante di Life è proprio in questo: non è possibile prevedere l'evoluzione di una data configurazione, bisogna costruirla materialmente.

Ed ecco che entra in ballo il computer. Nessuno avrebbe potuto studiare Life in modo approfondito senza il computer. Life è un gioco che non può essere analizzato per formule: va effettivamente giocato, e ciò è praticamente impossibile senza un calcolatore. Lo stesso Conway ne fece un uso massiccio per seguire l'evoluzione di qualche struttura particolarmente interessante, arrivando addirittura a calcolare qualche migliaio di mosse col PDP7 di cui disponeva. D'altronde è piuttosto semplice implementare un algoritmo di Life in praticamente qualsiasi linguaggio: la struttura cellulare si presta benissimo ad essere realizzata con una matrice di interi, positivi se la cellula è viva e nulli se è morta. Contare i vicini vivi per decidere lo stato di ogni cellula sono operazioni piuttosto semplici. Concettual-

mente, quindi, tutto bene. Vi sono però due problemi, uno di ordine teorico e l'altro di ordine pratico. Il primo è che il mondo dovrebbe essere infinito, cosa che chiaramente è impossibile a realizzarsi. A prescindere dalla quantità di memoria che si può dedicare alla matrice (più è, meglio è), il problema si aggira definendo adiacenti i bordi opposti della matrice, ossia facendo sì che le cellule della prima riga confinino (logicamente, beninteso) con quelle dell'ultima, e così per la prima e l'ultima colonna. In tal modo la matrice diventa illimitata pur rimanendo finita, come la superficie di una sfera. L'altro problema è il tempo di lavorazione, che aumenta col quadrato della dimensione della matrice. Contro questo ostacolo non si può fare nulla, a parte cambiare linguaggio o hardware, se non cercare di ottimizzare l'algoritmo velocizzando al massimo i calcoli più ripetitivi, quali ad esempio la determinazione del nuovo stato della cellula. Vediamo un modo piuttosto efficiente di farlo. Innanzitutto va tenuto presente che le possibili condizioni in cui una cellula si può trovare sono esattamente diciotto (infatti una cellula può avere da zero ad otto vicini, e per ognuna di queste nove evenienze può essere viva o morta). Ad ognuna di queste diciotto situazioni è associata la condizione futura della cellula, in base alle regole suesposte. Queste informazioni si possono facilmente organizzare in una tavola a diciotto entrate, la quale contenga il codice del nuovo stato in base alle condizioni attuali. (Esempio: se una cellula è viva ed ha zero vicini, muore; se è viva ed ha un vicino, muore; se è viva ed ha due vicini, vive; e così via per le altre quindici possibilità). A questa tavola si può accedere molto rapidamente se supponiamo che le cellule vive siano rappresentate con il valore 1 e quelle morte con zero. In questo caso basta moltiplicare per nove il valore di una cellula e quindi sommargli i valori dei suoi otto vicini; il risultato (compreso tra zero e diciassette) è l'indirizzo della tavola al quale si trova il nuovo stato della cellula.

Sempre rimanendo in tema di programmazione, un errore che si commette facilmente quando si programma Life per la prima volta è quello di usare una sola matrice per contenere la generazione vecchia e quella nuova. Bisogna usare due matrici, perché altrimenti la fase di aggiornamento non terrebbe più conto della situazione "congelata" alla generazione precedente, ma verrebbe inquinata dalla presenza della nuova generazione che si sta calcolando. Ricordiamo che la mutazione deve avvenire istantaneamente: per far questo bisogna necessariamente leggere da una matrice e crearne un'altra.

In definitiva si può dire che scrivere un programma di Life è piuttosto facile, ma ben più difficile è scrivere un buon programma di Life. A proposito: forse non tutti sanno che è possibile realizzare Life anche senza scrivere un vero e proprio programma, ma utilizzando un moderno spreadsheet tipo 1-2-3; chi volesse provarci scoprirà che si tratta di un passatempo piuttosto divertente ed istruttivo, e soprattutto meno facile del previsto. Saranno soprattutto le capacità logiche di programmazione ad essere messe alla prova: il compito implica infatti necessariamente il passaggio da un tipo di ragionamento algoritmico sequenziale ad uno in parallelo, al quale non siamo molto abituati.

Bene, per questo mese terminiamo qui, avendo visto sia la struttura e le regole del gioco che il modo in cui possono essere implementate in un semplice programma. Il mese prossimo presenteremo alcune configurazioni interessanti che si sviluppano da strutture piuttosto semplici e parleremo di come si possano apportare variazioni alle regole di base per ottenere classi di giochi simili a Life, ma dal comportamento differente. Speriamo di avere intanto suscitato il vostro interesse sull'argomento: vi consigliamo di provare per conto vostro a sviluppare Life o gli altri giochi di cui parleremo in futuro, e vi ricordiamo che potete scriverci in caso riteniate di avere qualcosa di interessante da proporci in merito. MC

Due diverse configurazioni di Life. Una è stata ottenuta da uno schema di partenza simmetrico, l'altra da uno casuale. Le immagini sono state realizzate da un programma per Apple inviato dal lettore Stefano Laporta di Bologna.

Life (II)

di Corrado Giustozzi

Eccoci nuovamente a parlare di Life. Nella puntata precedente di MCgiochi abbiamo visto, se vi ricordate, le regole di base di questo gioco affascinante. Abbiamo fatto conoscenza con gli strani abitanti unicellulari dell'infinito mondo bidimensionale di Life ed abbiamo visto le leggi che ne regolano nascita, morte ed evoluzione. Infine, abbiamo discusso brevemente di come si possano implementare in un programma per calcolatore le strutture e le regole del gioco. In questa seconda puntata ci addentreremo negli sviluppi del gioco stesso, cominciando col vedere due possibili strategie di approccio allo studio di Life e finendo con l'incontrare particolari configurazioni di cellule dall'evoluzione piuttosto peculiare e talvolta sorprendente.

Notiamo intanto che due sono le cose che viene spontaneo fare quando si ha a disposizione un (buon) programma di calcolo di Life: studiare l'evoluzione di una grande colonia di cellule ottenuta generando a caso gli individui, oppure esaminare il comportamento di ben precisi insiemi di poche cellule costruiti ad hoc. Entrambe le linee d'azione portano a sviluppi piuttosto interessanti, e perciò degni di essere discussi in questa sede; cominceremo pertanto questa puntata andando a vedere ciò che succede in ognuno di questi due casi. Nel primo, tutto sommato, serve ben poco: un "mondo" piuttosto vasto (almeno di 30 × 30) è l'unico requisito indispensabile. Si generano quindi a caso numero e posizione degli individui vivi al tempo zero, ottenendo la generazione iniziale; e poi... si sta a guardare. Già così il risultato sarà piuttosto interessante, ma sono possibili diversi miglioramenti. Un modo più raffinato di procedere potrebbe intanto essere quello di stabilire a priori la densità della nostra popolazione iniziale, espressa come rapporto percentuale fra numero di cellule vive e numero totale di cellule costituenti il mondo. Tanto per fare un esempio, se dispongo di un mondo di 30 × 30 cellule, partire con una densità del 25% significa che la mia generazione iniziale dovrà essere costituita da 225 individui. In questo modo si possono correlare densità iniziale e modalità di evoluzione della colonia, ottenendo chiari indizi di come sovrappopolazione o isolamento influiscano nel modificare la struttura nel corso del suo sviluppo. Potrebbe infine essere interessante disporre opportuni contatori nel programma, in modo che sul video compaiano di volta in volta indicazioni sullo stadio dell'evoluzione (generazione), sul numero di individui nati e morti nel passaggio dalla generazione precedente a quella attuale, sulla densità della popolazione attuale e sulla variazione rispetto alla generazione precedente. In questo modo si possono seguire gli eventi con maggior precisione, e si hanno più mezzi per interpretare il comportamento della colonia. I risultati che si ottengono mostrano che generalmente le colonie troppo dense o troppo sparse non hanno vita lunga, ma tendono ad estinguersi rapidamente o a degenerare piuttosto in fretta in strutture stabili, ossia che non mutano in numero e disposizione degli individui vivi da una generazione all'altra. Questo comportamento, del resto facilmente prevedibile, avviene in quanto in una generazione troppo poco densa sono poco probabili nuove nascite (che richiedono tre vicini vivi) mentre sono molto più probabili le morti per isolamento; al contrario, una generazione particolarmente densa ne produce immancabilmente una estremamente rarefatta, a causa delle numerose morti per sovrappopolazione. Aggregati con densità intermedie sviluppano invece comportamenti più variati e duraturi; la densità non varia di molto fra una generazione e l'altra, ed è probabile che col tempo si creino alcune "isole" stabili circondate da gruppi di cellule dall'evoluzione caotica ed incerta, il cui destino non appare chiaro se non dopo moltissime mosse.

Figura 1 - Un aliante, ossia una struttura che trasla sul piano al ritmo di una casella ogni quattro generazioni.

Figura 2 - Un piccolo mostro: nonostante la sua apparente semplicità la sua evoluzione è agitatissima ed il suo destino è ancora ignoto.

Appare evidente che questo modo di giocare Life permette, però, di comprenderne solo le proprietà di tipo statistico e per di più riferite alla globalità della struttura, mentre non ci mette affatto in grado di analizzare singoli comportamenti di gruppi di cellule, in particolare dal punto di vista geometrico. Per fare ciò è necessario poter disporre gli individui della generazione iniziale nel particolare modo voluto, compiendo cioè il secondo tipo di analisi di cui parlavamo in precedenza. Ciò richiede innanzitutto un programma un tantino più complicato del precedente: esso, ad esempio, deve permettere di descrivere la posizione di ogni individuo che si vuole in-

serire, ma deve nel contempo dare la possibilità di tornare indietro per variare la posizione di individui già inseriti, cancellandoli. Se ci si limita a strutture composte di pochi individui allora il mondo può essere anche di dimensioni piuttosto limitate, a tutto vantaggio della velocità di calcolo e dell'occupazione di memoria; ma vale sempre il principio per cui più spazio abbiamo a disposizione e meglio è. Tanto per esercitarci possiamo provare a vedere cosa succede con gruppi di tre individui adiacenti (appare chiaro dalle regole che gruppi aventi meno di tre membri muoiono subito). Provare per credere, delle cinque sole possibili disposizioni di tre cellule, tre si estinguono alla seconda generazione, una diventa subito un quadrato di quattro punti che rimane stabile, e l'ultima si trasforma i un ...oscillatore con periodo 2! La configurazione in questione è quella in cui le tre cellule sono allineate, ed alterna l'allineamento in direzione verticale a quello in direzione orizzontale. Con quattro cellule le cose non cambiano molto: le configurazioni possibili sono solo cinque, e di esse una (il quadrato) è e rimane stabile, tre si stabilizzano entro la terza generazione in una struttura vagamente esagonale detta "favo", ed una si trasforma alla nona generazione in un oscillatore di periodo due formato da quattro oscillatori a tre cellule del tipo visto prima. Non vi diciamo qual è la configurazione di partenza per non togliervi il gusto di scoprirlo da soli. Passando a gruppi di cinque individui le cose sembrano complicarsi parecchio, e si cominciano a fare incontri con oggetti piuttosto strani. Uno dei più interessanti è quello che Conway (l'ideatore di Life) ha chiamato "aliante" (fig. 1): la sua evoluzione è tale che ogni quattro generazioni il gruppo ritorna alla sua forma originaria, ma si trova spostato, nel piano, di una casella a destra ed una in basso! In altre parole, l'aliante trasla sul piano alla velocità di una casella ogni quattro generazioni. Se lasciato fare è in grado di percorrere tutto il mondo uscendo ben presto dallo schermo del vostro calcolatore, per poi eventualmente rientrarvi dal lato opposto se avete costruito il programma in base alle indicazioni date nella puntata precedente. Un comportamento piuttosto sorprendente, non vi pare? Un altro oggetto alquanto misterioso è quello che vedete in figura 2; potete provare a costruire il suo comportamento nelle prime mosse, ma sappiate che lo stesso Conway, col suo vecchio PDP-7, lo ha seguito fino alla quattrocentottantesima (!) generazione senza riuscire a capire se e come terminerà il suo intricato sviluppo. Se non avete tanta pazienza, ma volete comunque osservare qualcosa di simpatico, mettete cinque individui in fila indiana; anche una struttura semplice come questa vive una vita piuttosto complessa e movimentata!

Ma il passo definitivo nello studio di configurazioni particolari si compie quando si prendono in considerazione aggregati di una dozzina di individui o anche più. Le cose, adesso, si fanno veramente complicate, e servono decisamente un buon programma, un calcolatore veloce ed una notevole dose di pazienza (oltre a molta passione e molto tempo libero). Per fortuna gran parte del lavoro di scoperta è già stato fatto, evidentemente da qualcuno che disponeva di tutte queste cose. Anzi, da più di qualcuno: negli anni di grande boom di Life, un intero gruppo di ricercatori di intelligenza artificiale al M.I.T. si mise d'impegno allo studio di Life, ottenendo risultati che chiamare interessanti è poco. Per uno di essi, tra l'altro, vinsero addirittura un premio bandito dallo stesso Conway e destinato a chi avesse saputo confermare o confutare una sua congettura riguardante l'espansione delle strutture di Life. Conway pensava che nessuna struttura potesse accrescersi all'infinito, ossia generare un numero infinito di individui; il gruppo del M.I.T. riuscì a trovare un incredibile controesempio, che vedete in figura 3. Questa struttura di ventotto cellule, dall'apparenza innocua e "slegata", è invece un terribile "cannone ad alianti", come lo battezzarono i suoi scopritori. Nell'arco di quaranta generazioni esso si evolve in una struttura dall'aspetto altrettanto inverosimile, che costituisce il cannone vero e proprio: ossia un oscillatore di periodo 30, che "fabbrica" un aliante a cinque cellule (del tipo visto in precedenza) ad ogni oscillazione e poi lo lascia andare! Questa struttura è quindi in grado di dare vita ad un numero infinito di individui, i quali, oltretutto, se ne andranno in giro per l'eternità sull'illimitato piano di Life, sotto forma di aliante. Crediamo che il nome Life (Vita) per questo gioco sia, a questo punto, decisamente il più appropriato!

Bene, per questa volta terminiamo qui. Nella prossima puntata termineremo il discorso su Life... in bellezza, incontrando oggetti ancora più strani del cannone spara-alianti (ad esempio un mangia-alianti, che ne dite?). Vedremo pure qualche altro gioco della famiglia di Life, e discuteremo di come semplici variazioni alle regole di evoluzione portino a strutture e comportamenti assai diversi tra loro. MC

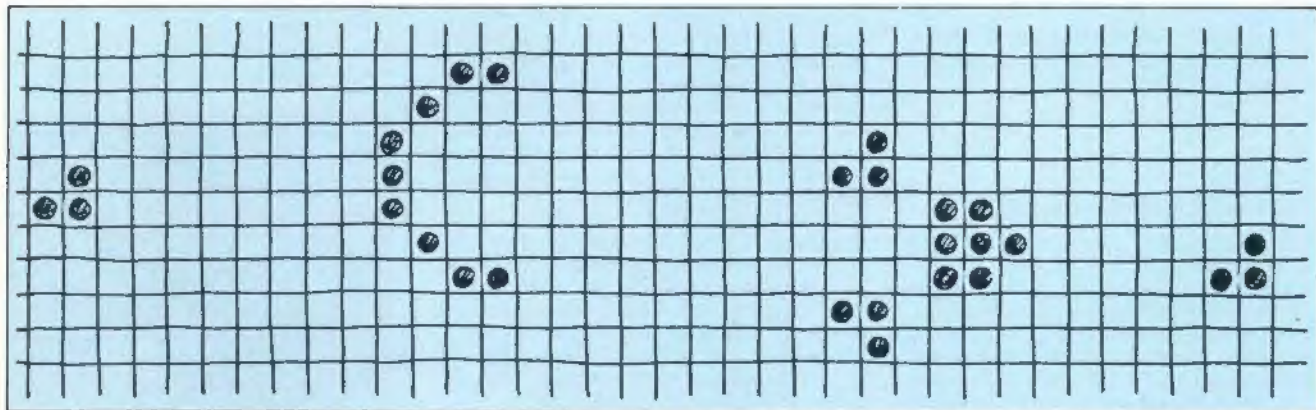
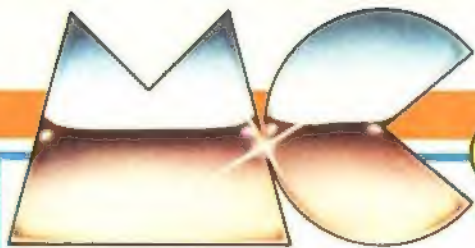



Figura 3 - Questo strano oggetto si evolve in una struttura in grado di "costruire" e "lanciare" alianti come quello di figura 1, al ritmo di uno ogni trenta generazioni, dando così vita ad un numero illimitato di individui.

Life (III)

di Corrado Giustozzi

Siamo quindi giunti al nostro terzo ed ultimo appuntamento con Life, l'avvincente gioco di simulazione al calcolatore inventato da J. H. Conway. Nelle puntate precedenti abbiamo visto in dettaglio scopi e regole del gioco e modi di implementazione su un personal, passando poi in rassegna alcune interessanti strutture dal comportamento piuttosto particolare. Questo mese termineremo il discorso facendo conoscenza con nuovi oggetti altrettanto peculiari, ed infine accenneremo a qualche estensione e/o variazione alle regole della simulazione che conducano a classi di giochi simili a Life nel concetto, ma diversi nel comportamento. La volta scorsa, se vi ricordate, parlammo di quel gruppo di ricercatori del M.I.T. autore di diverse scoperte sullo strano mondo di Life. Fra le creature da essi generate avevamo visto in particolare il mostruoso cannone ad alianti, che dà vita ad una infinità di individui che si spostano sul piano alla velocità di una cella ogni quattro generazioni. Bene, non contenti di ciò, i nostri scienziati misero a punto successivamente una struttura in grado di catturare e distruggere gli alianti emessi dal cannone! La vedete in figura 1; non lasciatevi ingannare dal suo aspetto semplice e lineare, ci troviamo di fronte ad un terribile mangiatore di alianti! (I suoi scopritori lo battezzarono pentadecatlon, in quanto si tratta di un oscillatore con periodo quindici). A seconda delle posizioni relative del pentadecatlon e dell'aliante, quest'ultimo può essere assorbito o, al contrario, venire riflesso e quindi "rimbalzare" all'indietro. Queste due possibilità possono essere entrambe sfruttate per ottenere costruzioni sorprendenti. Nel primo caso si può fare in modo che un pentadecatlon opportunamente collocato intercetti l'intero flusso di alianti emesso dal cannone, assorbendolo completamente; altrimenti si può creare una struttura surreale formata da due pentadecatlon che giocano a ping pong, rilanciandosi un aliante avanti e indietro.

Per la cronaca, esistono diversi tipi di alianti a cinque cellule; oltre a quello visto la volta scorsa potete osservarne un altro in figura 2. Il suo comportamento è esattamente analogo a quello dell'altro esemplare, ma questo tipo si differenzia dal precedente in quanto è alla sua razza che appartengono gli individui emessi dal cannone e mangiabili dal pentadecatlon.

Se a questo punto credete che questo sia il massimo e non sia possibile ottenere di più avete ben poca fiducia nei cervelloni del M.I.T.! Fra i virtuosismi di questo gruppo di Life-maniaci si annoverano oggetti che definire incredibili è poco. Per esempio, avreste mai pensato che fosse possibile collocare due o più cannoni ad alianti in modo che i vari individui in movimento giungessero ad una collisione tale che dall'interazione sorgesse una struttura a sua volta in grado di costruire e lanciare alianti di tipo diverso? Bene, anche ciò è stato fatto. Di questa struttura fantastica vi mostriamo solo l'ultimo stadio, ossia gli alianti prodotti; sono del tipo ad otto cellule riprodotto in figura 3, ed il gruppo del M.I.T. li ha denominati astronavi leggere per distinguerlo dal gruppo degli alianti a cinque cellule.

Sempre in tema di alianti: se ne mettete due come in figura 4, il risultato sarà la distruzione completa di entrambi: alla quarta mossa non rimane nessuna cellula viva. È possibile ottenere una cosa analoga per le astronavi? Sì, basta disporle come in figura 5; la morte totale avviene in questo caso alla settima mossa.

Come ormai già sapete, esistono in Life delle particolari configurazioni che non si evolvono affatto, ossia non mutano da una generazione

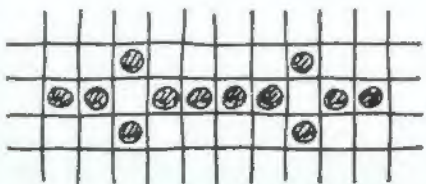
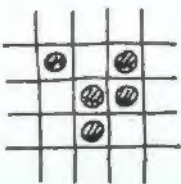
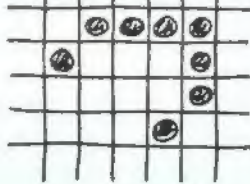
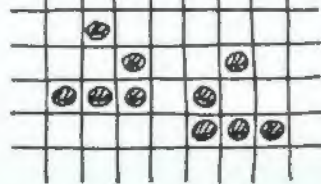



Figura 1 - Un terribile mangia-alianti, in grado di divorare un flusso di alianti come quelli di figura 2. Figura 2 - Un aliante a cinque cellule, parente di quello visto la volta scorsa. Sono di questo tipo gli alianti emessi dal cannone lancia-alianti. Figura 3 - Un'astronave leggera, ossia un aliante ad otto cellule. Figura 4 - Un cruento scontro di alianti: alla quarta generazione non vi sarà più nessuna cellula viva.

all'altra; esse sono dette stabili, e ne vedete due esempi in figura 6: il quadrato ed il favo. Una zona di mondo interamente costituita da ripetizioni di queste due strutture, purché le distanze reciproche non siano inferiori ad un minimo di "non interazione", costituisce un campo anch'esso stabile. Provate allora a costruire un simile campo e, una volta accertatane la stabilità, ad inserire uno ed un solo nuovo individuo in un punto qualunque della struttura. Cosa succede? Il nuovo arrivato crea senz'altro scompiglio nella configurazione, che comincia a perdere il suo equilibrio; poi, a seconda della geometria iniziale e della posizione dell'intruso, o tutto torna a posto e l'elemento perturbatore viene eliminato, oppure tutto l'ordine viene distrutto e la struttura originale si disfa nel caos. Ecco scoperto un interessante modello che rappresenta, ad esempio, l'aggressione di una coltura cellulare da parte di un virus: può darsi che la colonia attaccata sia in grado di sconfiggere l'intruso, ma più di frequente è quest'ultimo ad avere la meglio distruggendo l'intera colonia.

Abbiamo parlato più volte di oscillatori; un oscillatore nel mondo di Life altro non è che una configurazione avente la proprietà di tornare identica a se stessa dopo un certo numero di generazioni. In altre parole un oscillatore è una struttura la cui evoluzione consiste solo nell'assumere ciclicamente una medesima successione di forme e posizioni nel piano. Il suo periodo è il numero di generazioni dopo le quali ritorna alla configurazione di partenza. Il più semplice oscillatore l'abbiamo incontrato la volta scorsa: è formato da tre individui in linea retta ed ha periodo due (ossia è un flipflop). Il gruppo del M.I.T., fra le altre cose, ha costruito anche tutta una serie di oscillatori dai comportamenti complessi e strabilianti. In figura 7 ve ne presentiamo uno assai simpatico, detto "l'acrobata"; il suo numero consiste nel...fare le capriole, o meglio volteggiare attorno a se stesso, una volta ogni sette. E terminiamo con un piccolo capolavoro, detto Gatto del Cheshire. Chi ha letto Alice nel Paese delle Meraviglie conosce senz'altro il famoso Gatto del Cheshire, che svanisce lasciando visibile solo il sogghigno. Ebbene, in figura 8 vedete appunto il muso del Gatto (prima che svanisca, ovviamente). Provate a ricostruirne l'evoluzione. Avrete la sorpresa di vedere ben presto il muso dissolversi in configurazioni piacevolmente simmetriche che, alla sesta generazione, danno inaspettatamente luogo all'apparizione del sorriso; dopodiché anche questo sparisce ed al suo posto rimane, stabile, l'impronta della zampa del Gatto!

E con questa meraviglia finale giungiamo all'ultima parte del nostro viaggio nel mondo di Life. Resta da vedere come si possono estendere o modificare le regole del gioco per ottenere diverse varianti nel comportamento degli organismi in esame. Cominciamo col notare che esistono diverse categorie di regole applicabili ad una situazione "tipo Life". E per capire le possibili varianti appofondiamo un po' la questione. La regola di base è che lo stato futuro di una cellula deve essere stabilito in base a quello dei suoi vicini (eventualmente inclusa come caso particolare la cellula stessa). Già però la nozione di "stato" della cellula è suscettibile di ampliamenti. In Life esistono solo due stati: le cellule possono essere solo vive o morte, e le regole stabiliscono appunto quando ogni cellula debba cambiare di stato nascendo o morendo. Nulla impedisce però che le cellule possano assumere più di due stati; tre, ad esempio, che potremmo chiamare bianco, nero e grigio, o anche di più. In questo modo le regole possono consentire o impedire le possibili transizioni di stato in base ad eventi più complessi (anche i vicini hanno più stati, ovviamente, e questo complica le cose). E già che siamo in tema, notiamo che anche per il modo di scegliere i vicini si possono seguire due regole: considerare tutte e otto le cellule adiacenti a quella in esame (due in verticale, due in orizzontale e quattro sugli spigoli) oppure restringere l'attenzione alle sole quattro che confinano con essa mediante un lato (nord, sud, est e ovest). Nel primo caso, che è quello di Life, si dice che di ogni cellula si considera l'intorno di Moore, nel secondo caso l'intorno di Von Neumann, in onore ai due studiosi che hanno gettato le basi rigorose della teoria degli automi (e Life rappresenta proprio un esempio di automa cellulare, come dicemmo nell'introduzione a questa serie di articoli). Già vediamo che le cose sono diventate parecchio complesse. Ma ancora non è finito: rimane da scegliere il tipo di regole di transizione. Nell'ambito di ognuno dei due intorni si possono applicare diversi tipi di regole. Le due classi principali si chiamano totalistiche (o di conto) e geometriche. Esse stabiliscono lo stato della cella in base a quello dei vicini in modo rispettivamente "quantitativo" e "qualitativo". In base a queste considerazioni si possono costruire innumerevoli esempi di giochi tipo Life. Nella prossima puntata, quindi, ne vedremo qualcuno, entrando più in dettaglio nei criteri di scelta delle regole. MC

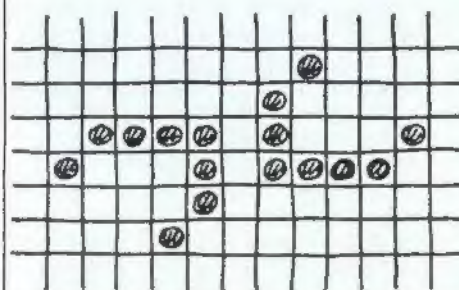
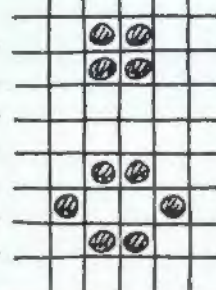
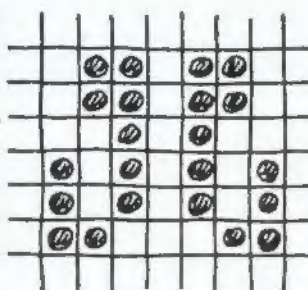
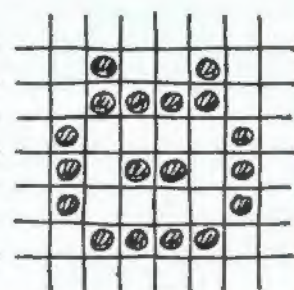



Figura 5 - Stesso scenario di figura 4, in cui però sono coinvolte due astronavi leggere. Figura 6 - Due figure stabili, ossia che non mutano da una generazione all'altra: sopra il quadrato, sotto il favo. Figura 7 - L'acrobata, che ogni sette generazioni si rovescia su se stesso. Figura 8 - Un capolavoro: il Gatto del Cheshire. Come quello di Alice, svanisce lasciando solo il sogghigno!

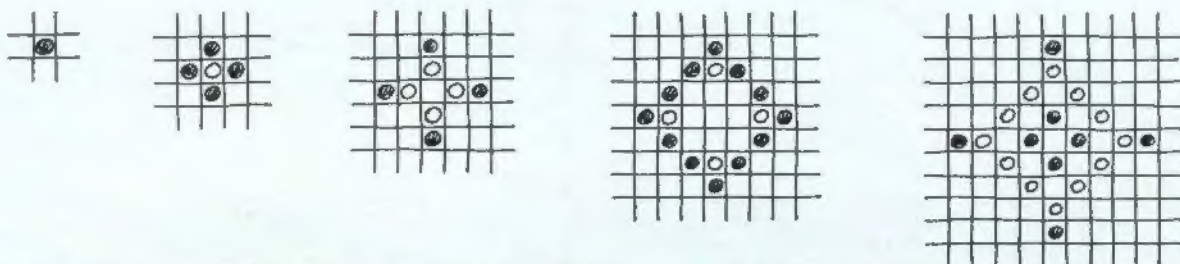
Life (IV)

di Corrado Giustozzi

Automi cellulari

Benché il nome della puntata di questo mese faccia pensare a cose da guerre stellari, l'argomento che ci accingiamo ad affrontare è in realtà la logica continuazione delle nostre precedenti esplorazioni del mondo di Life. Un automa cellulare è infatti un oggetto astratto formato da tanti elementi unitari elementari, detti cellule, ognuno dei quali può assumere uno o più stati in base al verificarsi di certi eventi; lo stato del complesso è quindi definito in base agli stati delle varie cellule. Da notare che l'automa non è il complesso di celle, ma ogni singola cella, in quanto vista come macchina a più stadi. Opportune regole stabiliscono tempi e modi delle transizioni di stato per ogni cellula, e quindi assegnano le leggi di evoluzione o comportamento dell'intero sistema. Se ci avete seguito nei tre mesi scorsi, a questo punto vi sarete certamente accorti che questa descrizione corrisponde perfettamente a Life. Ciò era stato accennato nella prima delle puntate dedicate a Life, e ricordato la volta scorsa, quando abbiamo rapidamente introdotto l'argomento delle estensioni alle regole del gioco per ottenere classi di giochi tipo Life, ma dai comportamenti diversi. Nella puntata di oggi riprenderemo questo discorso, ampliandolo grazie al concetto di automa cellulare, sempre per scoprirne i risvolti ludici.

Cosa sia un automa cellulare lo abbiamo appena detto, e nei mesi scorsi ne abbiamo anche conosciuto uno piuttosto da vicino; ma forse non abbiamo mai risposto alla domanda "ma a cosa serve?". Bene, non è certamente questa la sede opportuna per approfondire il discorso, ma in linea di massima possiamo dire che gli automi cellulari, essendo strutture formali astratte, permettono di definire proprietà o dimostrare teoremi di validità generale; essi svolgono praticamente la stessa funzione della Macchina di Turing, la quale, astratta ed irrealizzabile, serve però come base per stabilire principi assoluti di validità generale. La teoria degli automi cellulari è una branca piuttosto recente della teoria degli automi; due suoi pionieri sono stati Von Neumann e Ulam, che se ne sono occupati agli inizi degli anni cinquanta. Von Neumann, in particolare, ben si merita l'appellativo di papà di tutti gli informatici grazie ai suoi studi pionieristici in molti campi affini o coincidenti con ciò che oggi si chiama, appunto, informatica. Tornando agli automi cellulari, alcuni dei risultati stabiliti dalla teoria che li studia servono oggi di supporto ad altre moderne discipline: in primo luogo alla teoria dei sistemi, ma anche la ricerca operativa, l'intelligenza artificiale e la cibernetica, tanto per citarne qualcuna. Per quanto ci riguarda, poi, possiamo benissimo includere la ludica, o scienza dei giochi, ed in particolare la... ludomatica o informatica ricreativa, neologismo appena coniato con il quale intendiamo indicare la disciplina che si occupa di giochi intelligenti al calcolatore, come quelli trattati in queste pagine. Resi quindi gli... onori di casa, e fatta la debita conoscenza con gli oggetti in discussione, entriamo nel vivo del discorso, andando alla scoperta dei risvolti ricreativi degli automi cellulari. Possiamo iniziare ricollegandoci alla puntata precedente. Nell'ultima parte di quell'articolo suggerivamo qualche possibile strada per inventare nuove regole per Life, quali ampliare il concetto di stato o modificare le regole che tengono conto dei vicini. In questa puntata allora introdurremo organicamente i principali tipi di strutture e di regole che si possono adoperare per creare automi cellulari dal comportamento piuttosto interessante.

La struttura di fondo del gioco è sempre la stessa: un piano illimitato ed infinito, suddiviso da un reticolo in una infinità di celle uguali. Ogni cella è un automa cellulare, ossia una unità elementare del nostro sistema: può assumere certi stati ed intera-

Un modello di automa cellulare i cui stati dipendono anche dal tempo. L'evoluzione procede con una forte simmetria.

gire con le altre cellule. Lo stato di ogni cella può essere visualizzato come colore, o come un numero, o come si vuole; l'importante è che sia noto e non ambiguo, ed appartenga ad un insieme predefinito di stati possibili. Opportune regole stabiliscono quando ogni cellula debba cambiare di stato, e quale sia il nuovo stato da assumere; anch'esse ovviamente debbono essere non ambigue, ma oltre a ciò si richiede che siano perfettamente deterministiche, ossia non influenzate dal caso. La transizione di stato si effettua ad intervalli discreti di tempo, ed avviene istantaneamente e contemporaneamente per tutte le cellule interessate.

Questa struttura di base è comune a tutti gli automi cellulari; potremmo considerarla un po' come l'hardware dei nostri sistemi. Ciò che definisce il comportamento del sistema sono i due insiemi delle regole e degli stati ammissibili, che potremmo definire software dell'automa cellulare. Prima di discutere del software, però, possiamo soffermarci un attimo sulla struttura di base del sistema, osservando che qualche generalizzazione è possibile anche a questo livello. Il primo elemento su cui possiamo fermare la nostra attenzione è il reticolo stesso, ossia la struttura topologica dell'universo cellulare. Benché tradizionalmente si usi una suddivisione in celle quadrate, non è detto che questa sia l'unica alternativa possibile: reticoli isometrici andrebbero altrettanto bene, con celle quindi triangolari o esagonali come i tavolieri dei boardgame. La scelta del reticolo condiziona profondamente un'altra scelta, quella relativa all'intorno di applicazione delle regole di cui parleremo tra poco. Inoltre non è detto che il mondo dei nostri automi debba necessariamente essere bidimensionale: sarebbe altrettanto facile realizzarne uno tridimensionale, anche se forse sarebbe più scomodo lavorarci. In questi ultimi anni, al contrario, diversi studiosi hanno incentrato la propria attenzione su automi cellulari ad organizzazione unidimensionale. Benché a prima vista si possa pensare che ci sia poco da dire su di essi, invece queste strutture si sono dimostrate degne di interesse in quanto ricche di comportamenti non banali.

Stabilito comunque di dedicarci a reticoli quadrati bidimensionali, è opportuno prima di procedere, inquadrare il concetto di stato. Life, ormai lo sappiamo bene, è un gioco a due stati: ogni cellula può essere bianca o nera, o se preferite viva o morta. In generale però si può stabilire un insieme di stati formato da più di due elementi. Già con tre stati le possibili configurazioni di un certo insieme di cellule aumentano parecchio: n cellule a due stati possono assumere 2^n configurazioni, mentre se gli stati sono tre tale numero sale a 3^n. In generale se vi sono m cellule ognuna delle quali ammette n stati, il numero di configurazioni potenzialmente raggiungibili è m^n, enorme anche per moderati valori di n ed m. Naturalmente non è detto che ognuna di queste configurazioni possa effettivamente essere raggiunta in qualche momento dell'evoluzione; spetta alle regole stabilire il tipo di comportamento e quindi di evoluzione dell'automa: può essere che certe configurazioni siano impossibili in base alle regole date, e quindi non vengano mai raggiunte.

Se per gli stati c'è poco da dire, non così avviene per le regole di transizione; su di esse ci si può sbizzarrire a lungo, in quanto vi è veramente la massima libertà di scelta. Intendiamoci: non è assolutamente detto che la prima mezza dozzina di regole che venga in mente garantisca un automa dal comportamento duraturo ed interessante. Semmai è più probabile il contrario. Conway ha provato decine di varianti alle sue regole, prima di trovare quell'insieme che consentisse un delicato equilibrio di nascite e morti, e quindi un'evoluzione armonica ed interessante al suo Life. Però è anche vero che le regole potenzialmente applicabili sono moltissime, e solo una minima parte di esse è stata esplorata; e quindi è ancora possibile fare qualche scoperta interessante. L'assunto di base che deve essere rispettato da ogni insieme di regole di transizione è che lo stato di una cellula dipenda solo da quello delle altre celle di cui è formato l'automa. In altre parole non deve essere permesso riferirsi ad eventi estranei all'automa stesso, e quindi applicare regole del tipo "questa cella diventa nera se oggi è giovedì". Benché in teoria sia possibile far dipendere lo stato di ogni data cella da quelli di un qualsiasi altro numero di celle comunque disposte nell'universo (ivi compresa eventualmente la cella stessa), ciò che in genere si fa è considerare "condizionanti" solo quelle celle fisicamente adiacenti alla cella in esame. In particolare, quindi, ci si limita a considerare un intorno di ogni cella ogni volta che si deve valutare il suo nuovo stato. Ma anche con questa semplificazione, ci si trova di fronte a diverse alternative nell'operare la scelta; supponendo di operare su di un reticolo a base quadrata, ogni cella nel piano è circondata da esattamente altre otto compagne: due in verticale, due in orizzontale e quattro agli angoli. Quali di queste vanno considerate facenti parte dell'intorno? Naturalmente si può scegliere come si vuole, ma esistono due scelte che compaiono più frequentemente di altre nella letteratura. Van Neumann considerava adiacenti solo le quattro celle confinanti per un intero lato, mentre Moore includeva anche le quattro d'angolo; in loro onore adesso queste relazioni di vicinanza si chiamano rispettivamente intorno di Von Neumann ed intorno di Moore. Qualunque sia l'intorno considerato, esso si può dire proprio se non comprende la cella centrale, ossia quella di cui si deve decidere lo stato, e generalizzato in caso contrario.

Ma torniamo sulle regole di transizione di stato, e cominciamo col tentarne una semplice classificazione. Una prima classe di regole di transizione è formata dalle cosiddette regole di conto o totalistiche: esse stabiliscono lo stato della cella in base al numero di vicini aventi un certo stato, indipendentemente dalla loro posizione. Semplici sottoclassi di queste regole sono quelle definite di parità e di soglia. Le regole del primo sottogruppo sono applicabili principalmente nel caso di cellule a due stati, e sono del tipo "se il numero di vicini vivi è pari allora la cellula muore o rimane morta, altrimenti nasce o rimane viva". Benché molto semplice, questa regola produce insiemi di individui in grado di replicare la propria configu-

Questo modello, inventato da E. Friedkin, produce strutture che si replicano nel piano.

razione dopo un numero pari di mosse. (Vale forse la pena ricordare che Von Neumann era in origine interessato a studiare la possibilità di automi cellulari autoreplicanti). Le regole a soglia invece sono del tipo "se la cella ha almeno tre vicini vivi nasce, altrimenti muore", e, pur risultando di immediata applicazione nel caso bistabile, può essere generalizzata a situazioni con un maggior numero di stati.

Le regole di conto ignorano per definizione le relazioni geometriche esistenti fra le celle, e quindi costituiscono un insieme di leggi fortemente quantitative; il loro opposto sono le regole di tipo geometrico, che invece portano in conto la configurazione dell'intorno in esame. Una delle più semplici è del tipo "una cella nasce o rimane viva se il suo vicino di destra è vivo; muore o rimane morta in caso contrario", che produce un'evoluzione decisamente banale (sapreste dire quale?). Se ne possono però escogitare altre assai più complicate, che portino in conto anche gli stati dei vari vicini.

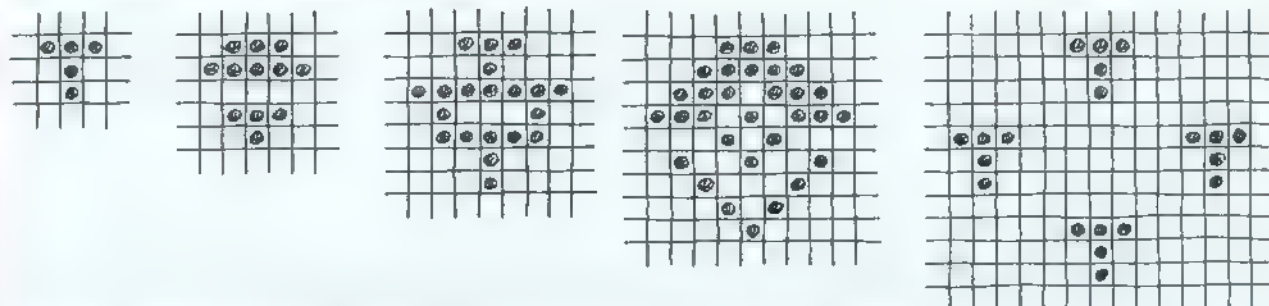
Un terzo gruppo di regole è formato da quelle che potremmo definire temporali o storiche, quelle cioè nelle quali conta anche il tempo, inteso come numero di "passi" (in Life le chiamavamo generazioni) trascorsi da un certo punto in poi. Ad esempio si può stabilire che ogni cellula rossa rimanga tale per non più di tre mosse e poi diventi verde, indipendentemente dallo stato dei suoi vicini. In questo modo le celle debbono conservare una "memoria" del loro passato, che concorre anch'esso a decidere il futuro della cellula stessa; facendo un parallelo con Life, le cellule soggette a questo tipo di regole possono quindi morire non solo per isolamento o sovrappopolazione, ma anche per... vecchiaia

Ed infine possiamo annoverare nel nostro elenco tutte le regole ibride formate dall'unione di più tipi visti precedentemente: leggi che siano contemporaneamente geometriche e storiche, o che magari siano ora dell'uno ora dell'altro tipo in funzione dello stato della cellula in esame

In questo modo si possono complicare le cose in maniera veramente perversa, ottenendo comportamenti piuttosto complessi e probabilmente difficilmente gestibili. Per concludere il discorso sulle regole, notiamo che anche in situazioni piuttosto semplici esse sono potenzialmente in numero così elevato da far paura; in effetti si può vedere che per le sole regole non storiche, se k è il numero di stati che ogni cella può assumere ed n il numero di celle da cui dipende l'evoluzione di ogni cella (il suo intorno, cioè), le possibili regole sono k^(k^n), un numero piuttosto grande. In pratica ciò si traduce nel dire che per celle bistabili nell'intorno di Von Neumann si possono definire ben 65.536 regole diverse, mentre in quello di Moore... fatevi voi il conto!

Giunti a questo punto potremmo considerare conclusa la puntata; non vogliamo però lasciarvi senza avervi dato almeno un paio di esempi di giochi da provare per rendervi conto del risultato. I due giochi sono volutamente molto semplici; crediamo che con quanto detto nel corso dell'articolo non abbiate difficoltà a modificarli ed ampliarli e ad inventarne addirittura di nuovi. A questo proposito vi segnaliamo che ci interesserebbe ricevere i risultati delle vostre sperimentazioni: già qualche lettore ci ha scritto a proposito di Life, se anche questo argomento dimostrerà di riscuotere il vostro interesse potremo ritornarci sopra in futuro, magari presentando i vostri contributi e le vostre idee

Per tornare ai giochi, eccovi due suggerimenti molto semplici, entrambi relativi a reticoli quadrati bidimensionali con intorno di Von Neumann. Il primo gioco sottostà a regole di tipo storico: in esso le cellule hanno tre stati, detti bianco, grigio e nero. Vi sono solo due regole, ossia: una cellula bianca diventa nera se e solo se confina con esattamente una cellula non bianca (non importa se nera o grigia); ogni cellula nera diventa grigia nella generazione successiva a quella in cui è diventata nera, e bianca in quella dopo ancora, indipendentemente dallo stato dei suoi vicini. Nel secondo gioco invece le cellule hanno due stati, che possiamo ancora chiamare bianco e nero, e l'unica regola è un'applicazione immediata della legge di parità; una cellula diventa o rimane bianca se i suoi vicini neri sono in numero pari, altrimenti diventa o rimane nera. Come vedete le regole sono in entrambi i casi semplicissime: i comportamenti che esse inducono negli organismi che le seguono, invece, lo sono un po' meno. Il primo gioco dà origine a delle interessanti configurazioni che, se l'insieme di partenza è simmetrico, mantengono un elevato grado di simmetria (provare il caso di un'unica cellula nera all'inizio). Il secondo, proposto da E. Friedkin del M.I.T., produce invece copie identiche della configurazione di partenza, realizzando così delle configurazioni autoriproducentisi.

Appuntamento al prossimo mese per intraprendere un argomento del tutto differente, ma sempre relativo alla classe dei giochi intelligenti al computer. MC

Un'altra forma del modello di Friedkin: il pentamino a T si riproduce fino ad invadere tutto lo spazio a disposizione

Passeggiando nel piano

di Corrado Giustozzi

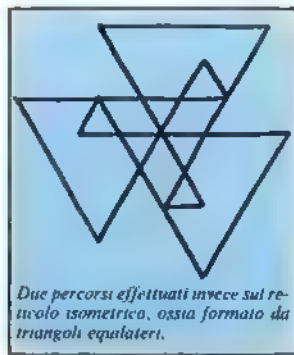
Immaginate per un attimo di trovarvi soli in un deserto, senza carta né bussola, il panorama intorno a voi è costante e monotono in ogni direzione, e non offre punti di riferimento o altri mezzi per fornirvi un orientamento. Cosa fate? Cominciate a camminare. In che direzione? Una a caso, tanto l'una vale l'altra. Questa descrizione può sembrare l'inizio di un Adventure Game ma è invece tutt'altra cosa; è un modo piuttosto colorito ma non per questo errato di descrivere la situazione in cui metteremo il nostro computer questo mese. Naturalmente non abbiamo intenzione di suggerirvi un bel lancio col paracadute assieme al vostro amato PC nel bel mezzo del Sahara: più semplicemente ci occuperemo di descrivere e quindi simulare il comportamento di un ente (persona, macchina, molecola) che si sposti in un piano illimitato in base a determinate leggi. Vedremo come si possono trovare interessanti situazioni analizzando queste "passeggiate" nel piano e scoprendone risvolti teorici affatto banali. Lo studio delle passeggiate nel piano nasce da diverse discipline: dalla chimica-fisica (si pensi al moto browniano) alla teoria dei sistemi, dalla teoria della complessità all'intelligenza artificiale. Naturalmente non sempre le passeggiate sono effettivamente tali: più spesso sono solo astrazioni di processi diversi che possono utilmente essere considerati come passeggiate, ossia spostamenti da un punto ad un altro di un piano o di un reticolo. In questa puntata ci occuperemo di diversi tipi di passeggiate nel piano: ognuna di esse può essere facilmente simulata con un calcolatore dotato di video grafico a media o alta risoluzione, meglio se un plotter, ed un programma che comprenda le più elementari nozioni di geometria. Tutte le passeggiate che vedremo avvengono in un piano illimitato ed infinito. Questo piano viene generalmente pensato come un reticolo a passo costante, e quindi le passeggiate non avvengono nel continuo ma ad intervalli discreti; naturalmente si assume come percorso unitario quello misurato dal lato del reticolo. La prima cosa che viene in mente di fare (e chi non l'ha mai fatto alzi la mano) è scrivere un programma di vagabondaggio casuale: si stabilisce un punto di partenza nel quale mettiamo il nostro vagabondo e cominciamo a gettare un dado a quattro facce. Se esce 1 spostiamo il nostro vagabondo di un passo verso l'alto, se esce 2 lo spostiamo di un passo verso est, se esce 3 verso sud e se infine esce 4 verso ovest. (Sorpresi per il dado a quattro facce? A parte il fatto che dadi tetraedrici esistono davvero e sono in vendita nei negozi di "giochi seri" ossia boardgame, potete anche ricorrere al generatore casuale del vostro computer). Se implementate un programmino del genere con tanto di grafica potrete ben presto deliziarvi alla vista di un puntolino luminoso che barcolla qua e là per lo schermo senza dare idea di dove voglia andare a finire. Cosa c'è d'interessante in tutto questo? Può sembrare una cosa da nulla, ma anche in questo semplice sistema si possono scovare problemi degni di nota. A che distanza dall'origine del percorso si troverà in media il vagabondo dopo n passi? Quanti vertici diversi avrà visitato? Che probabilità ha di tornare al punto di partenza? Forse non ve ne sarete accorti, ma questo confuso arrancare nel piano in funzione del lancio di un dado altro non è che un modello molto semplificato del moto browniano. Possiamo complicarlo passando dal reticolo al piano, ossia consentendo al nostro vagabondo di muoversi in tutte le direzioni e non solo verso i punti cardinali. In questo caso la soluzione ad alcuni dei problemi precedenti è nota: ad esempio quello della distanza media dall'origine. Ci ha pensato un ebreo svizzero a dimostrare, nel 1905, che il valore atteso della distanza del vagabondo dall'origine dopo n passi unitari è semplicemente 1 per la radice quadrata di n. Il suo nome? Albert Einstein.

Alcuni esempi di percorsi effettuati dalla tartaruga del Logo in un reticolo quadrato. Uno di essi è aperto, gli altri sono chiusi. Notare che le svolte avvengono sempre nel medesimo verso.

— Torniamo al nostro reticolo bidimensionale, e mettiamoci due vagabondi anziché uno solo. Se le posizioni di partenza sono scelte a caso, e ognuno dei due effettua una "passeggiata browniana", finiranno prima o poi per incontrarsi? La risposta dipende da come è fatto il reticolo. Se è finito e limitato da "barriere assorbenti" la risposta è *probabilmente no*. (Una barriera assorbente è un confine del reticolo. Se un viandante la tocca viene assorbito e la sua passeggiata ha termine) Se il reticolo è infinito ed illimitato allora la risposta diventa *certamente sì*! Anzi, se entrambe le passeggiate durano abbastanza a lungo (ossia per un tempo infinito) i due vagabondi si incontreranno certamente per un numero infinito di volte!

Come tutte le strutture matematiche, anche la passeggiata browniana in un reticolo piano è suscettibile di estensioni e generalizzazioni. Possiamo ad esempio pensare ad una passeggiata in un reticolo unidimensionale, ossia una retta graduata: il nostro vagabondo ora tira una moneta e quindi decide se fare un passo avanti oppure uno indietro. Dove conduce una simile mostruosità? Per esempio ad introdurre le "catene di Markov" e a mostrare interessanti agganci con la teoria della probabilità ed il gioco d'azzardo. Quale sarà la distanza attesa dall'origine del viaggiatore dopo *n* passi? Si dimostra essere circa 0,798 per la radice quadrata di *n*. Effettuate la vostra passeggiata casuale sulla tastiera di un pianoforte: partendo dal do centrale andate su o giù a seconda del lancio della moneta, suonando la nota su cui capitate. Com'è la musica risultante? Ritorneremo sulla generazione casuale di melodie in una delle prossime puntate, ed in quell'occasione capiremo anche perché questa "melodia" è così sgradevole da sentirsi.

Passiamo invece allo spazio, portando il nostro stoico vagabondo in un reticolo tridimensionale. Ora abbiamo bisogno di sei alternative, per cui un dado cubico fa al nostro scopo. Se il reticolo è finito è pressoché certo che il nostro viandante passerà prima o poi per qualsiasi vertice; e se è infinito? Ci ha pensato Georg Polya a dimostrare, nei primi anni venti, che anche con un tempo infinito a sua disposizione il viandante non ha la certezza di visitare ogni vertice.

Ma basta con le passeggiate browniane: passiamo dal perfettamente aleatorio al perfettamente deterministico. Supponiamo pertanto che il nostro viandante non sia più un vagabondo regolato dal caso ma un robot rigidamente governato da un programma. Cominciamo con un robot semplice, in grado di comprendere ed eseguire solo tre istruzioni: *avanti di un passo, gira a destra, gira a sinistra*, ed inoltre in grado di riconoscere l'eventualità in cui si ritrovi al punto di partenza. Il nostro primo programma è formato da tre istruzioni: 1) Avanti; 2) Destra; 3) Se sei tornato al punto di partenza allora fermati, altrimenti continua con il passo 1). Cosa fa il nostro robot quando obbedisce

Due percorsi effettuati invece sul reticolo isometrico, ossia formato da triangoli equilateri.

a questo programma? Ampliamo il nostro linguaggio di passeggio in modo che il robot capisca un'istruzione del tipo *avanti di n*, dove *n* è un intero. Cosa fa il seguente programma? 1) Avanti di 1; 2) Destra; 3) Avanti di 2; 4) Destra; 5) Avanti di 3; 6) Destra; 7) Avanti di 4; 8) Destra; 9) Avanti di 5; 10) Destra; 11) Se sei tornato al punto di partenza allora fermati, altrimenti continua con il passo 1).

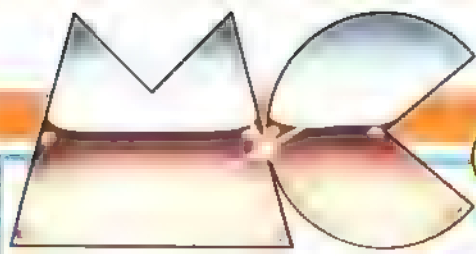
Questi simpatici programmini per robot passeggiatori sono stati escogitati da Seymour Papert, il noto ricercatore del MIT studioso di intelligenza artificiale. Questo robottino ambulante è stata una delle sue brillanti idee: l'ha chiamata *Turtle* (tartaruga) e l'ha incorporato nel linguaggio Logo, che almeno i più vecchi fra i lettori di MC dovrebbero conoscere bene.

Ma Papert ha fatto di più: ha elaborato uno splendido sistema che si evolve con incredibile complessità a partire da stati iniziali e regole semplicissime, senza che nessuno abbia capito *come* e *perché* vengano raggiunte certe situazioni. Un po' il Life della tartaruga, insomma. Al gioco (perché di questo dopodutto si tratta) è stata dato il nome *Worms* ossia Vermi. Funziona su di un piano illimitato ed infinito, suddiviso da un reticolo isometrico (ossia a triangoli equilateri). Si piazza il verme in un punto qualsiasi e lo si fa partire guidato da un programma. Il verme funziona come il robot e la tartaruga, solo che

Le scelte cui si trova di fronte il Verme per quanto riguarda i percorsi da seguire ad ogni vertice del reticolo.

divora il cammino che percorre in modo da lasciarsi dietro un solco che segna i tratti in cui è già passato. Il verme è vivo finché ha la possibilità di scavare solchi, ossia di percorrere cammini mai toccati prima: se per caso entra in un solco scavato in precedenza, muore. Naturalmente il linguaggio di programmazione del verme deve essere un po' più complesso di quello del robot, quanto meno perché il verme, procedendo su di un reticolo formato da rette che si incontrano a 60 gradi, si trova ad ogni vertice ben cinque strade possibili: due a sinistra (60 e 120 gradi), una avanti e due a destra (60 e 120 gradi). Se qualcuna di queste strade è già divorata, naturalmente il verme non può sceglierla (lo supponiamo esente da istinti suicidi) e quindi il suo campo di scelta sarà ridotto. Nel caso che tutte e cinque le alternative siano cunicoli allora il verme muore.

Osserviamo che in teoria con un programma composto al massimo da poche decine di istruzioni di scelta è possibile stabilire regole di comportamento per tutti i vermi in tutte le situazioni. In effetti le possibili configurazioni cui un verme si può trovare di fronte quando giunge in un vertice sono solo sei: cinque strade e zero cunicoli, quattro strade ed un cunicolo, tre strade e due cunicoli e così via fino a cinque cunicoli (caso mortale). Nel primo caso servono cinque alternative, ossia cinque regole di decisione, una per ogni caso (ammesso che si voglia essere così analitici), che possiamo ridurre a due sole unificando le due svolte a sinistra e le due svolte a destra, e scartando la strada centrale perché banale. Nel caso in cui ci sia un solo cunicolo, la scelta è tra quattro vie, e quindi servono quattro regole. Se i cunicoli sono due le scelte sono tre, e così via. Di ogni posizione possiamo anche considerare il numero di configurazione topologicamente diverse in cui si possono presentare le alternanze strada/cunicolo, ma anche così facendo non raggiungiamo le trenta scelte. Bene, questo insieme di regole, pur così limitato, conduce a situazioni incredibilmente complesse: percorsi agitati in cui simmetria e disordine si alternano, evoluzioni ordinate interrotte bruscamente da una morte inaspettata, percorsi convulsi che non si chiudono mai e proseguono all'infinito. Alcune simulazioni sono giunte ad analizzare qualche migliaio di mosse del verme, ma più si va avanti più l'apparente organizzazione di certe strutture scompare, lasciando lo spettatore col dubbio di cosa ci sia di casuale nell'ordine e viceversa.

Terminiamo qui per questo mese, lasciandovi alle prese col Verme divoratore; ma riprenderemo il discorso il prossimo mese.

Passeggiando nel piano (II)

di Corrado Giustozzi

Come già accennato la volta scorsa, anche per questo mese le nostre divagazioni ludico-matematiche saranno incentrate sul soggetto delle passeggiate nel piano. Parleremo di passeggiate ricorsive, e faremo la conoscenza con oggetti strani quali le curve del drago ed i fiocchi di neve.

Il mese scorso ci siamo occupati di passeggiate aleatorie e deterministiche. Nel discutere di queste ultime ci siamo trovati ad affrontare l'argomento dei "programmi di passeggio", ossia di quelle particolari successioni di istruzioni che descrivono un certo itinerario nel piano. Vogliamo adesso approfondire questo discorso per definire una nuova classe di passeggiate deterministiche generabili da speciali programmi di passeggio. Riprendendo quindi quanto già detto in precedenza, immaginiamo di utilizzare un ipotetico robot semovente (quale ad esempio la Tartaruga del linguaggio Logo) in grado di eseguire le istruzioni di uno di questi programmi; obbedendo ai comandi, il nostro robot traccerà sul piano un percorso che la volta scorsa abbiamo definito genericamente passeggiata. Come sarà fatto uno di questi percorsi dipende esclusivamente dal programma, ma per quanto riguarda la sua evoluzione possiamo in generale affermare che esso o finirà per chiudersi su se stesso oppure proseguirà all'infinito. Possiamo chiamare *percorsi chiusi* i primi e *percorsi aperti* i secondi, con ovvio significato dei termini. Nel primo caso avevamo supposto l'esistenza di una particolare istruzione di arresto per evitare al robot di continuare a girare per sempre in un percorso chiuso: essa semplicemente fa terminare la passeggiata nel caso in cui il robot si ritrovi in un certo momento nello stesso punto del piano dal quale era partito.

A questo punto possiamo tracciare una distinzione piuttosto netta tra i percorsi chiusi e quelli aperti, mentre i primi vengono completati in un tempo finito e racchiudono un'area finita del piano, i secondi non vengono mai completati (o, più precisamente, vengono completati in un tempo infinito) e delimitano un'area infinita del piano. Sembra plausibile che ogni percorso pensabile debba necessariamente appartenere ad una di queste due classi, ma vedremo fra un attimo che non è così.

Rivolgiamo quindi la nostra attenzione ai programmi di passeggio visti la volta scorsa: se vi ricordate, la loro struttura di base (sottaciuta, ma chiara) era sempre del tipo iterativo: *fai questa cosa finché non sei tornato al punto di partenza*, oppure anche *fai questa cosa per sempre* (iterazione infinita). In altre parole, ogni programma cui si è fatto cenno era sempre formato da una successione di istruzioni di movimento da ripetersi ciclicamente fino al verificarsi di un certo evento, od anche per sempre. Questi programmi in realtà definiscono esplicitamente il cammino che generano: la descrizione del percorso è contenuta nella successione di istruzioni del programma. Volendo complicare le cose si può pensare di scrivere un programma ricorsivo anziché iterativo; ciò equivale a definire il percorso in modo implicito ed in termini di se stesso, o meglio a descriverlo come limite di una successione di percorsi ognuno di quali è definito sulla base di quello che lo precede, e a sua volta definisce quello che lo segue. Un programma di passeggio siffatto dovrà naturalmente essere scritto in un linguaggio che ammetta la ricorsività, e quindi non in Basic.

Naturalmente il processo ricorsivo deve essere portato al limite per generare realmente il percorso, o meglio la curva che definisce implicitamente; e questo rende impossibile il percorrerlo in pratica. Si possono comunque prendere in esame le successive passeggiate generate ad ogni passo del procedimento, le quali, sebbene non godano delle proprietà del percorso limite, sono in effetti approssimazioni ad esso sempre più vicine; ciò permette di rendersi conto di cosa stia succedendo e di quali siano le proprietà di quest'ultimo. Il più semplice esempio di percorso esprimibile in forma ricorsiva è una curva che prende il nome da Hilbert: essa appartiene ad una vasta famiglia di mostruosità matematiche congetturate e poi dimostrate verso la fine dell'800 dal famoso matematico italiano Giuseppe Peano

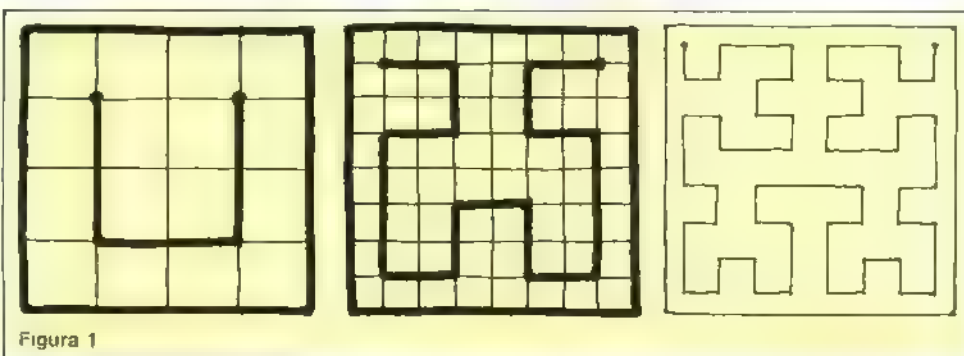



Figura 1

e note pertanto come *curve di Peano*. Queste sono strane entità che confusero non poco i matematici verso fine del secolo scorso, e portarono infine a modificare profondamente il significato del termine *curva* inteso nell'analisi. Il motivo per il quale queste "curve" crearono tanto scompiglio furono le loro proprietà quantomeno insolite: tanto per cominciare, una curva di Peano pur essendo una funzione continua non ammette un'unica tangente in ogni suo punto; nel nostro caso ciò equivale a dire che in nessun punto del cammino possiamo essere in grado di specificare in che direzione si sta muovendo il nostro robottino, nonostante che esso proceda lungo il percorso con continuità e con moto uniforme. Alcune fra le curve di Peano sono di lunghezza infinita pur racchiudendo un'area finita del piano; altre permettono al nostro camminatore di passare almeno una volta per ogni punto interno ad un quadrato in un tempo finito; altre circoscrivono un'area nulla pur essendo di lunghezza infinita e senza autointersezioni. Nella nostra ottica ci troviamo certamente ben lontani dalle tranquille passeggiate iterative viste la volta scorsa! Eppure anche le "passeggiate di Peano" sono ricche di interesse, ragion per cui andremo avanti nel loro esame.

Vediamo quindi in figura 1 i primi tre passi della costruzione di una curva di Hilbert: il procedimento ricorsivo appare chiaro, ma lo possiamo vedere esplicitato in maggiore dettaglio in figura 2. Il meccanismo è il seguente: si parte da un quadrato 4×4 contenente la curva iniziale (di *ordine zero*, come si dice),

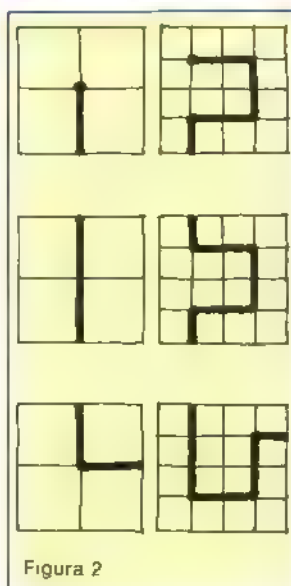



Figura 2

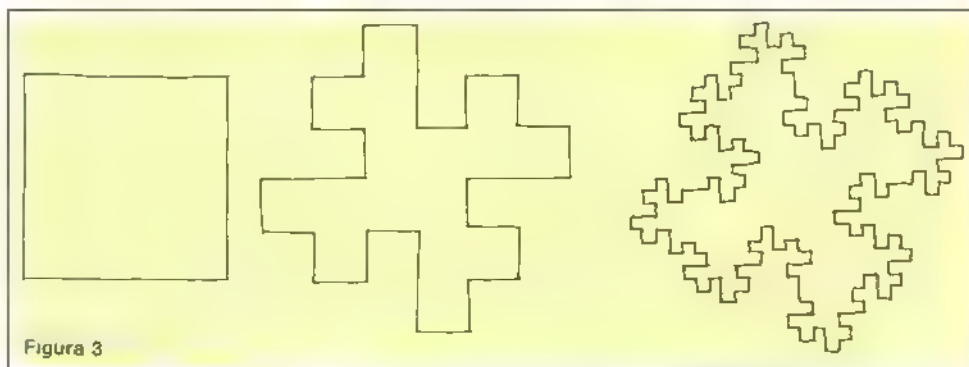
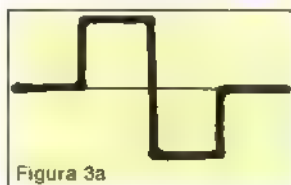



Figura 3a

ossia una specie di U. Si considerano quindi i sottoinsiemi della figura formati da quadrati 2×2, e si trasforma ognuno di questi sottoinsiemi nel modo specificato in figura 2. Il risultato globale è la curva di Hilbert di *ordine uno*. Applicando ricorsivamente lo stesso procedimento si va avanti a generare curve di ordine sempre maggiore. Ognuna di queste curve, lo ripetiamo, *non è* la curva di Peano; la successione così generata, tuttavia, *tende* alla curva di Peano, ossia si avvicina sempre di più ad essa. Notiamo come i due estremi della curva di Hilbert si spostino sempre più verso i vertici superiori del quadrato: in effetti la curva limite inizia nel vertice in alto a sinistra e termina, dopo un tragitto di lunghezza infinita, in quello in alto a destra. È facile anche vedere che la curva limite avrà lunghezza infinita: infatti dalla figura 2 si vede che il segmento di lunghezza unitaria diventa una spezzata di lunghezza 3,5 mentre il segmento e la spezzata lunghi 2 unità generano spezzate di lunghezza 4, per cui globalmente la lunghezza del percorso aumenta di un po' più del doppio nel passaggio da un ordine al successivo. Tuttavia l'intera curva rimane confinata in un'area quadrata del piano avente una superficie di 16 unità quadrate.

Col crescere dell'ordine aumenta la complessità della curva, misurata ad esempio dal numero di "svolte" ossia cambiamenti di direzione del nostro robottino ambulante. È chiaro come, nella curva limite, i cambiamenti di direzione avvengano con frequenza infinita: ed è questo in effetti il motivo per cui risulta impossibile specificare la direzione di percorrenza istantanea del robot, visto che cambia bruscamente infinite volte in ogni infinitesimo del percorso! Come accennato in precedenza, la curva di Hilbert non è la sola costruzione ricorsiva che generi una passeggiata di Peano. Possiamo quindi passare oltre, per vederne qualcun'altra di tipo diverso. Partiamo questa volta con una spezzata chiusa di quattro segmenti, in pratica un quadrato (figura 3). Ad ogni lato del quadrato applichiamo la trasformazione di figura 3a, che muta ogni segmento in una greca. Continuiamo quindi ad applicare la stessa trasformazione a tutti i segmenti della curva così generata, e procediamo ricorsivamente. In figura 3 vediamo i primi due ordini della curva generata, denominata "fiocco di neve quadrato" dal suo scopritore, il matematico polacco contemporaneo Benoit Mandelbrot. La curva naturalmente rimane chiusa e può essere considerata come il perimetro di una certa regione del piano. Notiamo ora una cosa interessante: nel

Figura 3

passaggio da un ordine al successivo la lunghezza della curva raddoppia, come è facile vedere; ma l'area da essa racchiusa non varia, in quanto la spezzata a greca "aggiunge" esattamente quanto toglie, con un risultato complessivo pari a zero. Ecco quindi che la curva limite, benché di lunghezza infinita, continua a circoscrivere un'area finita, esattamente pari a quella del quadrato iniziale! Prendiamo due punti qualsiasi sulla curva limite, e chiamiamoli A e B: quanto tempo impiegherà il nostro robot per andare da A a B se viaggia ad una velocità costante unitaria? La risposta ovviamente è che il nostro povero vagabondo dovrà camminare per l'eternità, in quanto anche un piccolo sottoinsieme della curva ha lunghezza infinita!

Le curve di Peano del tipo a "fiocco di neve" sono ben note, e ve ne sono di diverse specie. In figura 4 possiamo vedere uno degli esemplari più anziani, vero capostipite della razza, catturato nel lontano 1904 dallo svedese Helge von Koch. Si parte da un triangolo equilatero e si applica ricorsivamente la procedura consistente nell'applicare un nuovo triangolo equilatero nel terzo centrale di ogni segmento (4a). Anche in questo caso riportiamo in figura la relativa costruzione estesa fino all'ordine due. Come per la versione di Mandelbrot, anche il fiocco di neve di Koch al limite raggiunge una lunghezza infinita, pur racchiudendo un'area finita pari questa volta agli 8,5 di quella del triangolo iniziale. Le cose per il nostro robot non vanno molto meglio rispetto a prima; anche per i fiocchi di neve come per la curva di Hilbert vale il principio per cui non si può stabilirne l'orientamento nei vari punti del cammino.

Figura 4

Figura 4a

È piuttosto facile generare nuove curve di Peano, una volta afferratone il meccanismo; è anche piuttosto interessante studiarne l'andamento perlomeno per i primi quattro o cinque ordini. Potete provare ad inventare nuove curve generalizzando le regole del fiocco di neve: ad esempio costruendo più di un triangolo su ogni segmento, o costruendovi un quadrato anziché un triangolo. Oppure potete costruire i quadrati (o triangoli) verso l'interno anziché verso l'esterno; ciò dà origine ad una famiglia di curve ancora più strane, che possono anche autointersecarsi in più punti. Non sempre, comunque, si ottengono percorsi interessanti o esteticamente gradevoli; anzi, è piuttosto probabile trovarsi di fronte ad una curva bruttina e sgangherata, priva di particolare interesse. Un ottimo esercizio di programmazione è quello di scrivere un programma ricorsivo che generi una successione di curve di Peano, o che, dato l'ordine, disegni quella particolare curva. La cosa può essere fatta ad esempio in Pascal. Chi fra i nostri lettori è in grado di mandarci programmi che generino ricorsivamente curve di Hilbert o fiocchi di neve?

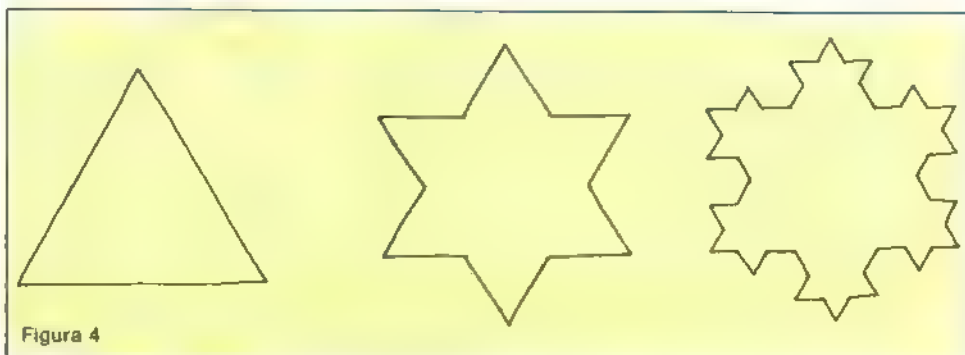
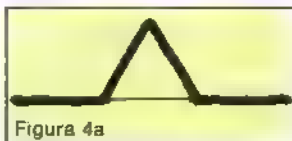
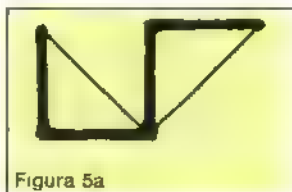



Figura 5a

E terminiamo con una delle ultime scoperte per quanto riguarda le curve di Peano, identificata meno di vent'anni fa da J. E. Heighway, un fisico della NASA, e da lui denominata "curva del Drago". Il procedimento ricorsivo che la genera è assai semplice, essendo basato sulla sostituzione di due segmenti formanti un angolo retto con quattro segmenti più corti disposti a due a due come i segmenti originari (fig. 5). Notiamo che la curva ha inizio e fine in due punti ben precisi, che non si spostano mai. Se ne supponiamo unitaria la distanza, vediamo facilmente che la curva di ordine zero è lunga 1,4142... unità (ossia radice di 2), e che questo incremento relativo di lunghezza viene mantenuto ad ogni incremento di ordine. Notiamo inoltre come la curva del drago abbia la tendenza ad autointersecarsi, anche se nei punti di contatto la curva non si incrocia con se stessa, ma rimane autotangente (ammesso che questo abbia qualche significato). Particolare curioso: le curve del drago sono state studiate a fondo da D. E. Knuth, meglio noto come autore della monumentale serie di volumi intitolata "The art of computer programming". In un suo articolo Knuth, dopo profonde dissertazioni matematiche sulle proprietà di queste curve, spiega anche come lui e sua moglie Jill abbiano decorato una parete della loro casa con una enorme curva del drago di ordine nove realizzata a mosaico con piastrelle in ceramica...

E con quest'ultima notiziola terminiamo la nostra passeggiata tra le passeggiate. Speriamo che vi siate divertiti con i nostri vagabondaggi e che non abbiate perso la bussola durante il cammino. Appuntamento quindi al prossimo mese, per un argomento ancora diverso. MC

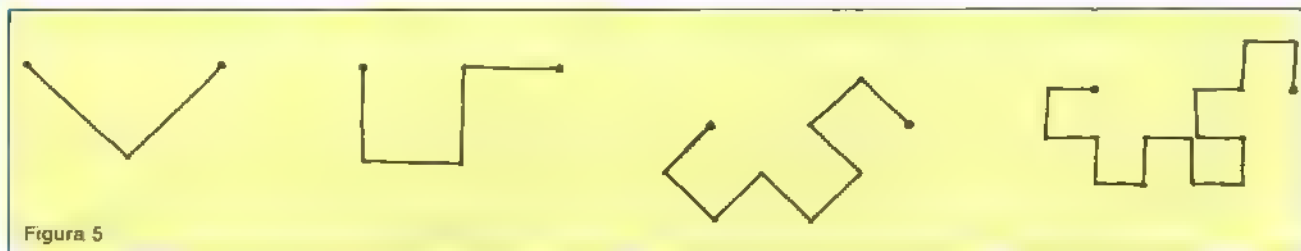
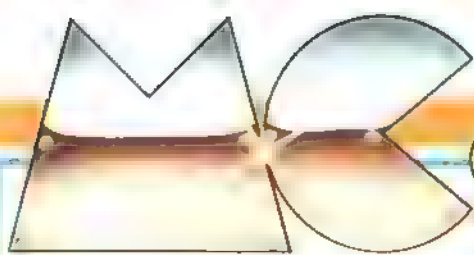



Figura 5

Problemi al calcolatore

di Corrado Giustozzi

Questo mese ci occuperemo dell'uso del calcolatore nella soluzione di particolari problemi di carattere matematico e di aspetto sia ricreativo che non.

Forse non ci capita spesso di pensare a quanto il calcolatore abbia modificato il nostro atteggiamento nei confronti della matematica: disponendo di un calcolatore è spesso assai più facile scrivere un programma che risolva un certo problema con la "forza bruta" piuttosto che mettersi con carta e matita a ricercare la soluzione teorica. Un esempio? Eccovene tre: trovare un quadrato perfetto di quattro cifre che scritto da destra a sinistra rimanga un quadrato; trovare un quadrato perfetto di quattro cifre nel quale le prime due cifre siano uguali tra loro e così le seconde; trovare un numero che sia uguale alla somma delle cifre del suo cubo. Questi problemini algebrici hanno oltre un secolo di vita, e le loro soluzioni possono essere ricavate col ragionamento: ma quanti di noi al giorno d'oggi si dedicherebbero a ricercarle per via teorica? Molto più facile dar di piglio al Basic, tirando giù una dozzina di righe al volo che produrranno i risultati in una manciata di secondi.

La stessa facilità con cui il calcolatore permette di risolvere questi semplici quesiti può essere sfruttata per ricercare le soluzioni di problemi di cui non si conosca completamente la teoria. Pensiamo al famoso problema delle otto regine: proposto oltre un secolo e mezzo fa fu affrontato e risolto da Gauss e da altri per via teorica, ma solo per il caso particolare della scacchiera di lato otto. È facile dimostrare che il problema è generalizzabile, ossia che è sempre possibile disporre n regine su di una scacchiera $n \times n$ in modo che non ve ne siano due sotto attacco reciproco. Però fino ad oggi nessuno è stato in grado di trovare la formula che dia il numero di soluzioni dato l'ordine, ossia il lato della scacchiera. Questo è uno di quei casi in cui il calcolatore è l'unico mezzo per sondare il problema, inaffrontabile con carta e matita. Ed in effetti grazie al calcolatore oggi si conoscono le soluzioni fino all'ordine sedici, e un po' meno per la variante detta delle *super-regine* o delle *amazzoni*, ossia in cui contano anche gli attacchi a salto di cavallo. (Per la cronaca una delle prossime puntate di MCgiochi sarà interamente dedicata al problema delle regine ed alle sue numerose varianti.)

Applicare il calcolatore per ricercare soluzioni di vecchi indovinelli può sembrare frivolo, ma spesso questo tipo di ricerche porta ad interessanti scoperte teoriche. E comunque è esattamente in questo modo che nel 1976 un gruppo di ricercatori americani è riuscito a risolvere un "indovinello" che aveva tormentato i migliori matematici per 124 anni: la famosissima *congettura di Guthrie* o *problema dei quattro colori* ora più precisamente denominata *teorema dei quattro colori*. Tutto cominciò nel 1852 quando un certo Francis

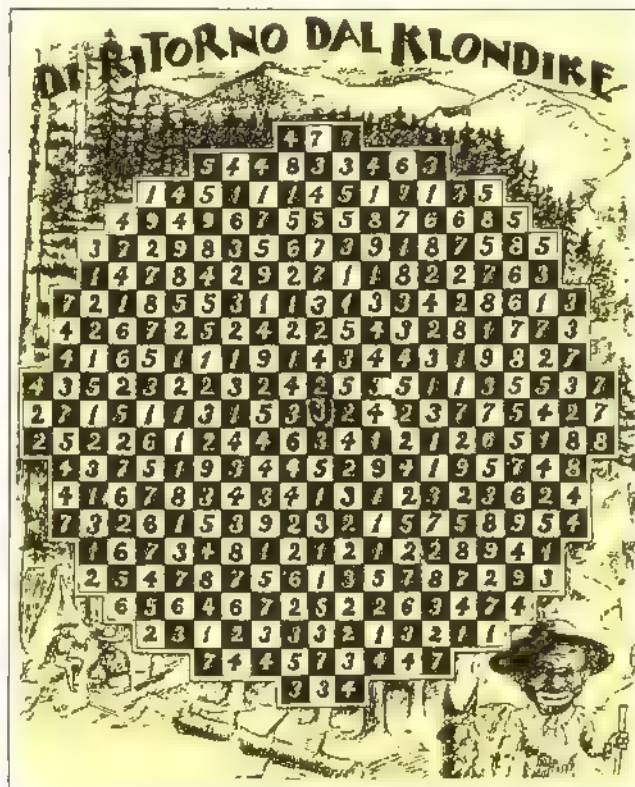



Figura 1 - Il gioco "Ritorno dal Klondike" di Sam Loyd (da "Passatempi Matematici" vol. I, Sansoni 1980).

Guthrie scrisse a suo fratello Frederick, all'epoca allievo del celebre matematico Augustus de Morgan all'University College di Londra, di un problema che si era posto e che riguardava la colorazione delle carte geografiche. Egli supponeva che qualunque mappa disegnata su un foglio di carta potesse essere colorata con quattro colori soli senza che per questo due diversi paesi confinanti venissero ad avere lo stesso colore. In altre parole egli supponeva che quattro colori bastassero per colorare qualsiasi mappa in modo che ogni paese non avesse lo stesso colore di quelli con esso confinanti. Egli chiedeva al fratello se fosse stato a conoscenza di una dimostrazione matematica di questa sua congettura. Frederick non la conosceva e pertanto riferì la questione a de Morgan, il quale però non ne sapeva più di lui. In effetti il problema non era mai stato posto prima di allora, e de Morgan cominciò a lavorarci sopra, riuscendo però solo a dimostrare che cinque colori erano in ogni caso sufficienti a colorare qualsiasi mappa rispettando la condizione. Che esistessero mappe per cui quattro colori fossero necessari era accertato: ma esisteva almeno una mappa che richiedesse necessariamente cinque colori, o quattro erano in ogni caso sufficienti? De Morgan non riuscì a stabilirlo. Il problema fu portato alla notorietà nel 1878, quando fu presentato alla London Mathematical Society nientemeno che da Arthur Cayley il quale al pari di de Morgan non era riuscito né a dimostrarlo né a confutarlo. Da allora il problema è rimasto indeciso per quasi un secolo e un quarto; tutti i matematici hanno sempre ritenuto che la congettura di Guthrie fosse vera, e quindi che quattro colori fossero sufficienti oltre che necessari, ma nessuno era mai riuscito a dimostrare che fosse veramente così oppure a mostrarne la falsità presentando un controesempio di una mappa che richiedesse cinque colori. Questo almeno fino al 1976, quando la congettura è stata dimostrata vera in un modo che nessuno avrebbe mai immaginato: mediante quattro anni di studi, due di preparazione teorica ed oltre un migliaio di ore macchina di calcoli sull'elaboratore dell'Università dell'Illinois a Urbana. È questo il primo esempio nella storia di dimostrazione matematica non ripetibile da chiunque con carta e matita; un precedente che ha diviso i matematici moderni in due parti, da un lato i sostenitori delle dimostrazioni tradizionali e dall'altro quelli che accordano piena validità a "dimostrazioni" di questo tipo.

Non crediate però che il calcolatore venga adoperato solo per indagare o risolvere quesiti che nascondono profondi significati matematici o teorie non del tutto svelate: spesso anche vecchi passatempi vengono sottoposti al vaglio dei programmi di calcolo per scoprirne i lati deboli. È questo il caso di un rompicapo ideato da Sam Lloyd, il famoso enigmista e scacchista americano vissuto a cavallo dei due secoli. Pubblicato per la prima volta nel 1907 sotto il nome di "Ritorno dal Klondyke", il gioco (v. fig. 1) consiste nell'uscire dal reticolo numerato partendo dalla casella centrale a forma di cuore; la regola per fare ciò consiste nello scegliere una direzione (Nord, Est, Sud, Ovest o una diagonale) e spostarsi di un numero di caselle pari alla cifra contenuta nella casella in cui ci si trova, ripetendo il movimento (scegliendo ad ogni passo una nuova direzione) finché non ci si trovi al di fuori della griglia. Sam Lloyd nelle risposte ai giochi affermava che esisteva un'unica soluzione, ossia una sola sequenza di passi per uscire dal reticolo. Ma questo è stato dimostrato falso nel 1977, quando un programma Fortran trovò qualche centinaio di percorsi diversi che portavano all'esterno. I programmatori riuscirono a dimostrare che ciò era dovuto ad una svista del disegnatore: tutti i percorsi alternativi passano infatti per una stessa casella non compresa nella soluzione originale di Lloyd e contenente un due. Bene, sostituendo al due qualsiasi cifra tranne il sette continuano ad esistere soluzioni alternative; solo mettendoci il sette le cose tornano, e non sono possibili altre vie di uscita che quella segnalata da Lloyd. Appare quindi verosimile l'ipotesi dell'errore di trascrizione, del quale non si accorse neppure lo stesso Lloyd.

Un altro dei problemi che periodicamente impegnano i più grossi computer del mondo è quello del calcolo delle cifre di pi greco e della ricerca dei più grandi numeri primi. Per quanto ci risulta, attualmente pi greco è stato calcolato fino alla milionesima cifra decimale, mentre il numero primo più alto conosciuto è $(2^{132049})-1$, un numero di oltre 40.000 cifre! Esso come si vede è della forma $(2^n)-1$: questo tipo di numeri primi vengono detti *primi di Mersenne*, in onore del matematico francese che per primo li studiò a fondo, nel 1700. Essi costituiscono un sottoinsieme dei numeri primi dalle proprietà piuttosto interessanti. La loro caratteristica principale è quella di essere intimamente legati con un altro insieme di numeri piuttosto particolare, i cosiddetti *numeri perfetti* noti e studiati sin dall'epoca di Platone. Ma andiamo con ordine.

Innanzitutto i primi di Mersenne sono molto pochi,

n	Primo di Mersenne (2^n)-1	Perfetto (2^(n-1))*((2^n)-1)
2	3	6
3	7	28
5	31	496
7	127	8.128
13	8.191	33.550.336

Figura 2 - I primi cinque numeri perfetti, derivati dai primi cinque primi di Mersenne

1	220	284
2	1.184	1.210
3	2.620	2.924
4	5.020	5.564
5	6.232	6.368

Figura 3 - Le prime cinque coppie di numeri amicabili

	Numero	Lunghezza
1	12.496	5
2	14.316	28
3	1.264.460	4
4	2.115.324	4
5	2.784.580	4

Figura 4 - I numeri iniziali delle prime cinque catene di numeri socievoli

tanto che la scoperta di uno di essi viene considerata un avvenimento. Attualmente ne sono noti ventotto, e l'ultimo della lista è quello citato poc'anzi. Perché sono così pochi i primi di Mersenne? Un motivo è che è necessario che l'esponente di 2 sia primo perché anche il numero risultante lo sia; ciò non è però sufficiente, come si vede con semplici controesempi. Ad esempio la cosa non funziona con 11 come esponente, in quanto $(2^{11})-1$ fa 2047 che è il prodotto di 23 per 89. Fra l'altro non si sa ancora se i primi di Mersenne siano in numero finito od infinito, una questione piuttosto importante per la teoria dei numeri. Cosa c'entrano i primi di Mersenne con i numeri perfetti? Bene, ricordiamo prima cos'è un numero perfetto: è un numero che gode della proprietà di essere uguale alla somma dei suoi divisori. Ad esempio il 28, che è divisibile per 1, 2, 4, 7 e 14; e 1+2+4+7+14 fa appunto 28. Il più piccolo numero perfetto è 6, il più grande attualmente noto è $(2^{132048}) * ((2^{132049})-1)$. Ecco il collegamento con i primi di Mersenne: se $(2^n)-1$ è un primo di Mersenne allora $(2^{(n-1)}) * ((2^n)-1)$ è un numero perfetto pari. Questo fatto fu notato da Euclide, ma fu dimostrato rigorosamente solo da Eulero, che provò che questa formula è in grado di produrre tutti i numeri perfetti pari. Finora nessuno ha mai trovato un numero perfetto dispari, e nemmeno si sa se numeri di questo genere possano esistere o no. Indagini al calcolatore hanno dimostrato che non possono esservi perfetti dispari inferiori a 10^{36}, ma nessuno sa cosa possa succedere al di là di questo limite inferiore. Cosa ci si fa con i numeri perfetti? Assolutamente nulla, almeno non molto di più che con le ultime 999 000 cifre di pi greco. Ma si sa, i matematici sono gente strana, e vanno sempre alla ricerca di cose strane e difficili per il solo gusto di farlo, riuscendo anche a trovarci una soddisfazione di natura estetica. Come nel caso dei numeri *amicabili*, anch'esso imparentato ai numeri perfetti. Due numeri si dicono amicabili se la somma dei divisori del primo è uguale al secondo e viceversa. I pitagorici conoscevano il caso della coppia 220 e 284, che consideravano simbolo di amicizia. In effetti una relazione così simmetrica fra due numeri è piuttosto bella e rara, tanto che una seconda coppia fu scoperta solo nel 1636 da Pierre de Fermat (17 296 e 18 416). Cartesio ed Eulero ne scoprirono altre, e così Legendre. In questi ultimi anni molte ore macchina sono state spese un po' in tutto il mondo per trovare nuovi numeri perfetti da aggiungere all'elenco; attualmente dovremmo essere vicino alle 2000 coppie, le più grandi delle quali hanno qualche centinaio di cifre. Nel 1918 il matematico francese Poulet annunciò di avere trovato un'estensione piuttosto notevole alla relazione di amicabilità, che egli chiamò *socievolezza*. I numeri *socievoli* sono una catena in cui la somma dei divisori di uno di essi è uguale al successivo, e così di seguito fino a tornare al primo. Poulet annunciò due catene del genere: la prima di 5 elementi (cominciante con 12.496) la seconda di ben 28 elementi (cominciante da 14.316), che è tuttora la più lunga che si conosca. Tentativi di ricerca esaustiva al calcolatore di catene socievoli sono stati fatti in più parti fra cui il M.I.T.: il risultato è che ora si conoscono un paio di dozzine di catene fra le quali nessuna di tre soli numeri, fatto questo che ancora non ha trovato una spiegazione teorica.

Questi sono solo alcuni dei casi in cui il calcolatore ha svolto o svolge un ruolo primario (se non decisivo) nella ricerca teorica; se volete si tratta sempre di risolvere degli indovinelli, di dimensioni più o meno grosse, ma sempre indovinelli. Che differenza c'è fra cercare il prossimo primo di Mersenne e la prossima configurazione del problema delle regine, o fra l'analisi del Ritorno dal Klondike e quella delle mappe a quattro colori? Cos'è poi gioco e cos'è ricerca?

Ma non vorremmo buttarla troppo in filosofia della scienza. Per cui terminiamo qui, augurandovi buone vacanze. Dopo la pausa estiva ci ritroveremo con una rubrica rinnovata e piuttosto diversa, come avrete letto nel riquadro, in cui affronteremo sempre problemi di aspetto ricreativo, ma in modo più pratico, quasi sperimentale. Attendiamo fin d'ora i vostri pareri in merito e nel frattempo vi diamo un arrivederci a settembre. MC

MCgiochi cambia

Dal prossimo numero di settembre la rubrica MCgiochi cambia aspetto e collocazione. Da semplice esposizione degli aspetti teorici di argomenti più o meno ricreativi di matematica o informatica diventerà un luogo di incontro e sperimentazione sostanzialmente pratico, una palestra per lo scambio di esperienze fra tutti i lettori che amano divertirsi col calcolatore in modo crativo ed intelligente. Almeno a giudicare dal feedback che abbiamo ricevuto in questi mesi crediamo che questo sia quello che la maggior parte di voi desidera. Ecco quindi che dalla prossima volta la rubrica abbandonerà la sua collocazione tra i videogiochi e comincerà ad occuparsi di argomenti diversi e soprattutto in modo differente da quanto avvenuto finora.

In programma a breve scadenza abbiamo diverse cose interessanti: dalla presentazione della nostra versione di Core Wars ad un po' di sperimentazione sulla sintesi melodica pseudocausale, da una discussione sul problema delle regine ad una puntata sulla simulazione e manipolazione del linguaggio, ad una sui programmi di simulazione di semplici ecosistemi. Naturalmente i vostri interventi sono non solo benvenuti, ma anche richiesti: nell'ottica di affrontare le cose in modo operativo contiamo di pubblicare anche e soprattutto listati di programmi relativi agli argomenti trattati. Potete anche segnalarci eventuali argomenti che volete vedere affrontati in maggior dettaglio, e magari inviarci i vostri lavori. Qualcuno di voi già l'ha fatto o lo sta facendo: il primo che ha meritato una pubblicazione ante litteram è stato Walter Tross, il cui ottimo programma di simulazione Wa-Tor viene per questo numero ospitato nel software Apple.

Avanti allora, contiamo di ricevere i vostri interventi su qualcuno degli argomenti citati in precedenza, o su altri a vostra scelta. Avete tutta l'estate davanti per dare sfogo alla vostra creatività ed inviarci i risultati. A rileggerci dunque a settembre nella nuova rubrica ex-MCgiochi.

Nel numero scorso vi avevamo accennato che nella rubrica MCgiochi, o più esattamente nell'articolo di due o tre pagine che ogni mese la apre, ci sarebbe stato qualche cambiamento. Non radicale, come vedrete: semplicemente, visto che da parecchi mesi ci stiamo occupando di "giochi intelligenti" abbiamo preferito dare a questa rubrica una collocazione più precisa, più a sé stante, piuttosto che lasciarla come semplice introduzione alle recensioni di programmi di giochi in commercio.
Avevamo detto che avremmo avuto piacere che questo spazio diventasse anche un luogo di scambio di esperienze fra i lettori che amano divertirsi con il calcolatore in modo creativo e intelligente. Tanto per stimolarvi, questo mese, cominciamo con... un bel raccontino, mentre aspettiamo i vostri contributi.
Ovviamente, non raccontateci solo cosa avete fatto ma, nei limiti del possibile, anche come.
m.m.

Io e ... le Regine

di Corrado Giustozzi

Se credete che sia facile mettere otto regine su una scacchiera in modo che non si mangino, significa che non ci avete mai provato...

Il problema delle otto regine è un argomento antico che però ogni tanto ritorna alla ribalta. Spesso però chi lo cita non lo conosce bene o ne ignora l'origine, e lo riporta in modo errato o incompleto, specie per quanto concerne numero e tipo delle soluzioni. Peccato, perché il problema in sé non è privo di interesse, come vedremo: inoltre è ricco di generalizzazioni e presenta risvolti informatici piuttosto significativi.

Come forse sarà noto a molti di voi,il problema si situa a metà strada fra la matematica e gli scacchi; la sua storia è piuttosto lunga, e nasce proprio su di una rivista scacchistica che si pubblicava a Berlino verso la metà del secolo scorso. Sul numero di settembre 1848 della *Schachzeitung*, un tale Max Bezzel proponeva come rompicapo di disporre sulla scacchiera otto regine in modo che non ve ne fosse nessuna sotto attacco da parte di qualcun altra.

Ciò in pratica equivale a richiedere di collocarle in modo che non ve ne siano mai più di due sulla stessa riga o colonna o diagonale.
A costo di rovinarvi la sorpresa vi dico subito che il problema ammette dodici soluzioni distinte, senza contare le rotazioni e le riflessioni. Volendo contare anche queste ultime (cosa che però di solito non si fa) il loro numero sale a novantadue. (Non a novantasei, come qualcuno avrebbe potuto pensare, in quanto una soluzione base è simmetrica e non genera altre sette soluzioni, ma solo altre tre). Ad ogni modo il quesito proposto da Bezzel, subito denominato *problema delle otto regine*, ottenne presto una grande fama: fu tra l'altro affrontato nonché risolto anche da Gauss, il quale come si sa non perdeva molto tempo in stupidaggini. Però la prima soluzione pubblicata fu quella di Franz Nauck, che apparve nel 1850 sulla *Illustrierte Zeitung* di Lipsia: fra l'altro essa correttamente citava le soluzioni senza contarne riflessioni e/o rotazioni. Gauss al contrario aveva determinato tutte le soluzioni, senza eliminare quelle ottenute per rotazione o riflessione dalle altre. Una dimostrazione che le soluzioni di Nauck erano tutte e sole quelle possibili fu infine pubblicata dal matematico inglese J.W. Glaisher dell'università di Cambridge, ed apparve sul numero di dicembre del *Philosophical Magazine*. Glaisher fu inoltre il primo ad estendere il problema a scacchiere non standard, ossia con un numero di caselle per lato (detto *ordine*) diverso da otto: in particolare egli fornì le corrette soluzioni per le scacchiere di ordine cinque e sei. Nello stesso anno il matematico tedesco Gunther pubblicava un suo metodo basato sui determinanti per mezzo del quale si poteva trovare in linea di principio una soluzione al problema originario, ma in effetti al massimo lo si poteva applicare in pratica per l'ordine sei in quanto all'aumentare del lato della scacchiera il numero dei calcoli da svolgere cresceva in modo astronomico

Cosa ne è ora del problema delle regine? Diciamo che viene considerato come una curiosità matematica di un certo interesse, anche se priva (sembra) di risvolti pratici. Ciò sembrerebbe liquidare la questione, se non fosse per un piccolo particolare seccante: nessuno è ancora riuscito a dare una formula che leghi l'ordine della scacchiera al numero delle soluzioni. L'unico modo per sapere quante sono le soluzioni al problema per un dato ordine è ancora quello di andare a trovarle una per una, compito, naturalmente, che il pigro uomo moderno lascia volentieri al calcolatore. Esistono diversi algoritmi noti per fare questo, che si dividono in due grosse fasce: quelli che in fondo non fanno altro che ripetere ciò che un uomo farebbe metodicamente a mano e quelli che

Le dodici soluzioni base al problema delle otto regine, nell'ordine in cui vengono isolate dal programma di ricerca

procedono per vie traverse, risolvendo problemi puramente numerici che si sa essere isomorfi al problema originale In ogni caso il numero dei calcoli da eseguire cresce in modo così vertiginoso al crescere dell'ordine della scacchiera che le soluzioni sono note per pochi ordini oltre quello originario, e fa ancora notizia l'aggiunta di un nuovo ordine superiore da parte di qualche supercomputer sparso per il mondo. Notiamo invece che, al contrario, è abbastanza agevole dimostrare analiticamente che esiste almeno una soluzione al problema per ogni lato maggiore di tre ed è assai facile vedere che non si possono inserire più di *n* regine su di una scacchiera di lato *n* senza violare i termini del problema

Si sa come i matematici, quando hanno tra le mani un problema interessante, non resistono all'impulso di generalizzarlo; il problema delle regine non è un'eccezione, ed è dovuto sottostare anche lui a diverse estensioni. La prima cosa che è venuta in mente, non si sa più nemmeno a chi, è stata quella di sostituire alla regina altri pezzi degli scacchi. Con i pezzi ortodossi, però, il problema si fa poco interessante: *n* torri, ovviamente, possono essere disposte in modo da non attaccarsi in esattamente n! modi (n fattoriale), e questo conclude la questione (anche se nessuno si è mai preso la briga di isolare le soluzioni base neppure per gli ordini più bassi). Le cose non migliorano passando agli alfieri, ed assumono aspetti addirittura grotteschi con i cavalli: penso che tutti sappiate che su una normale scacchiera è possibile disporre trentadue cavalli (in un unico modo) in modo che non ve ne siano mai due sotto attacco reciproco! Dobbiamo quindi rivolgerci agli scacchi eterodossi: il pezzo maggiormente candidato a risollevare le sorti della vicenda è la cosiddetta *amazzone* o super-regina, che possiede contemporaneamente le proprietà di movimento della regina e del cavallo. Con essa le cose tornano a diventare interessanti: è noto che il *problema delle amazzoni* non ha soluzione per scacchiere di lato inferiore a dieci, e ne ha esattamene una (o quattro, contando le riflessioni) per l'ordine dieci. Tuttavia anch'esso rimane elusivo come quello delle regine, essendo ancora irrisolta la questione se esista una formula precisa che correli ordine della scacchiera e numero delle soluzioni, e in caso quale sia. Anche per questo caso esiste un esercito di supercomputer sguinzagliati sulla terra che ogni tanto provvedono ad aggiornare il limite superiore delle nostre conoscenze: però succede generalmente che ogni nuovo risultato calcolato contraddica tutte le congetture fatte in precedenza, cosa piuttosto sintomatica della nostra scarsa comprensione teorica del problema.

Due interessanti minimi a confronto. A sinistra l'unica soluzione al problema delle regine di ordine 4. A destra l'unica soluzione a quello delle amazzoni di ordine 10. In entrambi i casi non esistono soluzioni per gli ordini inferiori, e le soluzioni sono uniche (salvo rotazioni e riflessioni).

Il mio primo incontro/scontro col problema delle regine avvenne quando avevo circa otto anni: lo lessi da qualche parte e provai a risolverlo manualmente, spostando otto pedine su di una scacchiera vera Dopo diversi tentativi casuali e piuttosto goffi, giunsi a determinare una specie di procedura "prova e riprova" che, posto di avere avuto abbastanza pazienza, mi avrebbe permesso di trovare tutte le soluzioni Ne trovai effettivamente qualcuna poi mi stufai: a quel pun-

to il problema era, come suol dirsi, virtualmente risolto, e me ne dimenticai presto. Naturalmente a quell'epoca l'idea di usare un calcolatore non mi sfiorava neppure l'anticamera del cervello. Il problema mi tornò in mente invece parecchi anni dopo, quando divenni felice possessore di una delle prime TI-59. Spesi diversi giorni, a più riprese, per riuscire a far entrare un programma decente in quei dannati quattrocento passi di memoria; la terza versione del programma, la più efficiente, trovò correttamente le novantadue soluzioni in poco più di quaranta ore di elaborazione (!). Un secondo programma si occupò poi di leggere le soluzioni trovate ed analizzarle al fine di eliminare quelle ottenute per rotazione e o riflessione dalle altre. Cinque o sei ore furono sufficienti per isolare correttamente le dodici soluzioni base, con grande gioia della mia TI-59 che vedeva finalmente il termine di quella prova-supplizio. L'algoritmo che avevo implementato era piuttosto contorto, ma aveva il duplice vantaggio di essere molto compatto e di usare poca memoria, fattori entrambi vitali considerato l'hardware a mia disposizione: faceva uso di un solo vettore di otto posizioni, e si basava sulle proprietà aritmetiche delle coppie di numeri estratte da questo vettore. A questo programma, che ancora posseggo, è legato un piccolo aneddoto. A quel tempo bazzicavo la redazione di una neonata rivista romana di computer. (no, non MC; un'altra, precedente) direttore della quale era (indovinate un po'?) Paolo Nuti, e coordinatore l'immancabile Marco Marinacci. A Marco piacque l'idea strampalata del programma che aveva girato per cinquanta ore su una programmabile per risolvere un problema inutile, e mi chiese di scriverci sopra un articolo; la rivista era fresca fresca, aveva bisogno di materiale e d'altronde io ero già stato ospitato su quelle pagine, anche se in modo poco esteso. Morale, scrissi l'articolo; e siccome piacque finì che ne scrissi un altro, e poi un altro, e poi... il

Ordine	Soluzioni Generalizzate	Soluzioni Base
4	2	1
5	10	2
6	4	1
7	40	6
8	92	12
9	352	46
10	724	92
11	2.680	341
12	14.200	1.787

Figura 1 - Il numero di soluzioni al problema delle regine dall'ordine 4 all'ordine 12 Notare che gli ordini 4 e 6 ammettono una soluzione unica

Ordine	Iterazioni Ricerca	Iterazioni Isolamento
4	28	2
5	86	39
6	296	7
7	1 022	170
8	3 928	414
9	16 082	1 606
10	69 628	3 330
11	328.490	12.345
12	1 683 976	65.607

Figura 2 - La complessità della ricerca e dell'isolamento delle soluzioni del problema delle regine cresce in modo esponenziale, come si può vedere dai valori qui riportati Essi rappresentano il numero di volte in cui il programma ha richiamato la routine centrale dell'algoritmo, ossia quella del piazzamento di una nuova regina nel caso della ricerca e quella del controllo di esistenza precedente di una rotazione-riflessione nel caso dell'isolamento delle soluzioni fondamentali

seguito lo immaginate da soli.

Ma torniamo al problema delle regine. In seguito ebbi occasione di lavorare con un medio sistema (Honeywell Livello 62) dotato di un buon compilatore Fortran 77, e mi tornò alla mente il vecchio problema. A questo punto non c'erano problemi di memoria e potevo dedicarmi solo all'efficienza del codice. Scrissi quattro versioni successive del programma, e l'ultima, la più ottimizzata, differiva in velocità dalla prima per circa un fattore trenta. Comunque anche così non riuscii ad andare oltre l'ordine 12, che impegnò la macchina per circa 11 ore in una ricerca esaustiva di tutte le 14200 soluzioni generalizzate, con estrazione delle 1787 soluzioni base nonché 22 *supersoluzioni* base, ossia soluzioni al problema delle amazzoni. Da estrapolazioni stimai che il tempo per la ricerca completa di ordine tredici si sarebbe aggirato sulle centoquaranta ore, un tantino oltre il ragionevole considerato anche che nella letteratura era noto il numero di soluzioni fino all'ordine 18 Ma tanto per non dargliela vinta (al problema), mi misi a cercare strade alternative, quali quella di analizzare solo il problema delle amazzoni. Ciò non è peggio come può sembrare; in effetti la maggior rigidità nella selezione fa diminuire il numero di configurazioni da esaminare e quindi accellera il calcolo. Inoltre adottai un metodo di ricerca non esaustiva delle soluzioni generalizzate, che forniva un miglioramento di circa un fattore due alla velocità di ricerca. In questo modo arrivai fino all'ordine quattordici, un risultato che non ho trovato nella letteratura. Ma questo era veramente il capolinea, almeno con quel calcolatore.

Ma il tempo passa, la tecnologia progredisce, e così dopo un ulteriore pausa di molti mesi ho finito per riaffrontare il problema, questa volta con l'XT che si trova sul mio tavolo di lavoro. Incredibile ma vero, gli stessi programmi che giravano sul 62, ricompilati per l'XT col Fortran 77 Microsoft, hanno girato più velocemente che sul vecchio mastodonte! E ciò mi ha permesso di raggiungere l'ordine 15 in circa otto ore di ricerca (ed un paio previste di isolamento che però non ho più effettuato). E per ora questo è tutto. Da qualche tempo però il pensiero mi è ritornato, e sto meditando, nei pochi momenti liberi, di riscrivere tutto in C per vedere che succede: ma per fortuna non l'ho ancora fatto e non credo che lo farò. Un'altra cosa che ho fatto solo parzialmente è l'analisi matematica dei risultati delle varie ricerche, in quanto a complessità di calcolo, tempi di esecuzione, numero di soluzioni trovate o isolate. Un argomento abbastanza interessante, del quale non mi pare sia stato pubblicato nulla I miei programmi alla fine di ogni ricerca provvedevano a stampare un rapportino contenente i valori di alcuni contatori di eventi critici nonché i tempi di calcolo Ho messo tutti i risultati in uno spreadsheet e ho fatto qualche regressione, ma i risultati non sono affatto chiari, tranne che la complessità di calcolo del mio algoritmo sembra essere più che polinomiale.

Ordine	Soluzioni Base
10	1
11	6
12	22
13	239
14	653

Figura 3 - Il numero di soluzioni al problema delle amazzoni dall'ordine 10 all'ordine 14

E con questo concludiamo questo breve excursus nel problema delle regine. Nelle tabelle pubblicate ho riportato il numero di soluzioni generalizzate e di quelle base (con tanto di complessità di calcolo) per il problema delle regine dagli ordini da quattro a dodici, e quello delle soluzioni base per il problema delle amazzoni dall'ordine dieci al quattordici. Chi fosse interessato ad ulteriori notizie ed eventualmente ai listati dei programmi Fortran di ricerca può senz'altro scrivermi presso la redazione

In due puntate della precedente versione di questa rubrica, per la precisione quelle di maggio e giugno scorsi, avevo parlato dell'argomento "passeggiate nel piano". La prima parte era dedicata alle passeggiate "semplici" e parlava di passeggiate casuali, della tartaruga del Logo, di un giochino topologico detto Worms e di percorsi chiusi ed aperti; la seconda trattava invece delle passeggiate "ricorsive", ossia di quel tipo di percorsi definiti "di Peano" o "Hilbertiani", dal nome dei matematici che per primi li hanno studiati. Questo tipo di percorsi è piuttosto particolare in quanto può essere generato solo facendo uso di una procedura ricorsiva, tale cioè che chiami se stessa nel suo corso. Le curve generate in questo modo godono di proprietà piuttosto strane, alcune delle quali erano state descritte nel testo. Diversi lettori sono stati stimolati dall'argomento ed hanno sviluppato programmi ricorsivi di calcolo e tracciamento di curve di Peano; rispondendo quindi all'invito contenuto nell'articolo hanno pensato di inviarli in redazione. Questa seconda puntata di MCintelliGIOCHI sarà quindi dedicata all'esposizione dei loro lavori, che come vedremo sono tutti piuttosto interessanti.

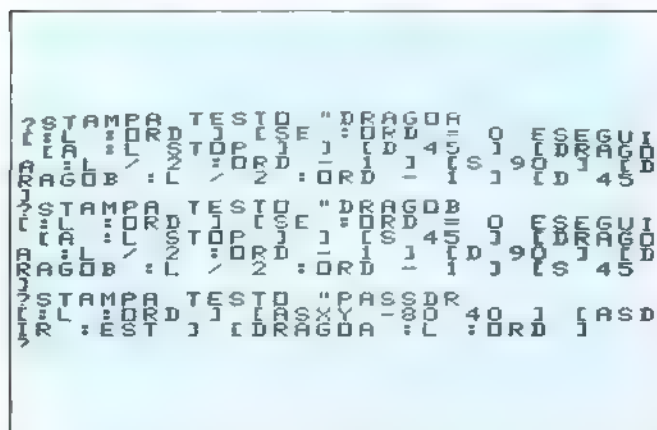
Le passeggiate dei lettori

di Corrado Giustozzi

La tartaruga sulla curva del drago ...

Nel citato articolo sulle passeggiate ricorsive (Passeggiando nel piano II, in MC n. 42 giugno 85, pag 40) accennavo alla possibilità di scrivere programmi in Pascal che generassero curve di Peano; in effetti è abbastanza classico che i produttori di compilatori Pascal forniscano nel pacchetto di demo per il loro prodotto anche un programma in grado di tracciare una curva di Hilbert del tipo di quella pubblicata in figura 1 dell'articolo citato. Altri linguaggi moderni permettono la ricorsività, ma non pensavo che fossero molto diffusi qui da noi finché non mi sono visto arrivare programmi per passeggiate ricorsive scritti in Logo e addirittura in Lisp! Segno evidente che diversi utenti "creativi" hanno (finalmente!) abbandonato il Basic per passare a qualche altro linguaggio più remunerativo.

Fra i vari interventi ricevuti credo che il più interessante sia quello di Piero Fiozzo, uno studente di informatica di Mestre che ha usato il ben noto TI Logo II su di un Texas TI 99 4-A. Le sue sperimentazioni lo hanno portato a scegliere di tracciare curve aperte e non chiuse (come i fiocchi di neve) per le limitazioni imposte dalla macchina alla risoluzione sullo schermo. La scelta del Logo come linguaggio mi sembra

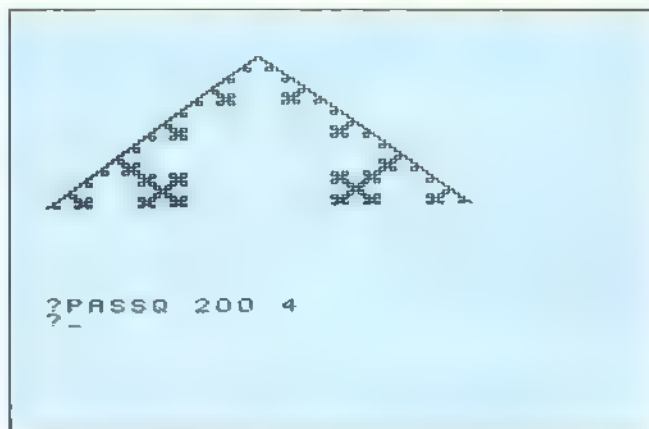
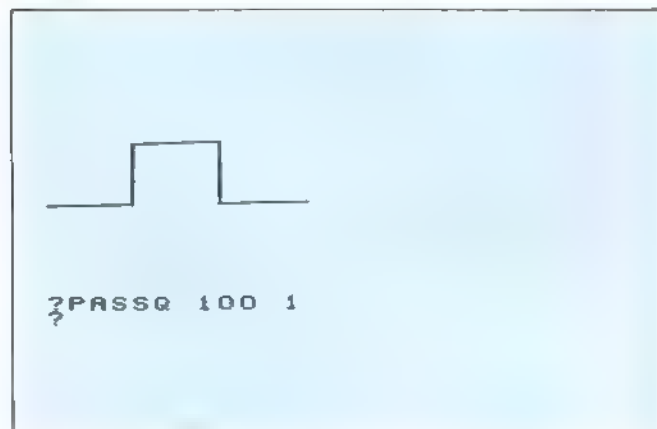
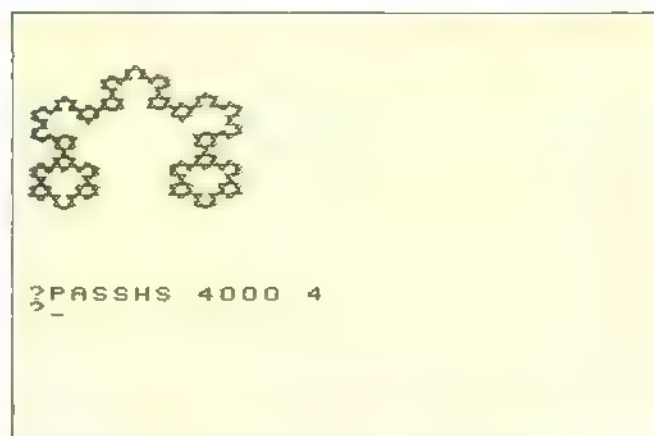
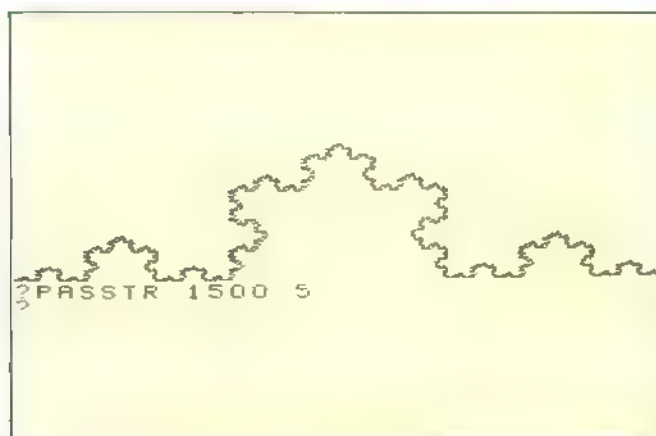
```
?STAMPA TESTO "DRAGOA
[:L :ORD ] [SE :ORD = 0 ESEGUI
 [A :L STOP ] ] [D 45 ] [DRAGO
A :L / 2 :ORD - 1 ] [S 90 ] [D
RAGOB :L / 2 :ORD - 1 ] [D 45
]
?STAMPA TESTO "DRAGOB
[:L :ORD ] [SE :ORD = 0 ESEGUI
 [A :L STOP ] ] [S 45 ] [DRAGO
A :L / 2 :ORD - 1 ] [D 90 ] [D
RAGOB :L / 2 :ORD - 1 ] [S 45
]
?STAMPA TESTO "PASSDR
[:L :ORD ] [ASXY -80 40 ] [ASD
IR :EST ] [DRAGOA :L :ORD ]
?
```

Incredibile ma vero, le poche righe di sinistra generano la curva di destra, una curva del drago di ordine nove. Il programma è scritto in TI Logo per il TI-99 4-A.

piuttosto degna di nota, soprattutto alla luce della concisione del codice realizzato, tutti i programmi inviati (sei, per la precisione) constano di pochissime istruzioni, cosa alquanto stupefacente per chi invece è abituato alla prolissità del Pascal. I vari programmi eseguono alcune trasformazioni ricorsive del tipo di quelle illustrate nell'articolo, oltre a qualche variazione sul tema tipo l'anticurva della trasformazione quadrata 'classica' e una trasformazione a base esagonale. Pezzo forte è però quello che calcola le curve del drago, realizzato per mezzo di due procedure che si chiamano ricorsivamente l'una con l'altra. Per mezzo di esso il nostro lettore è riuscito a tracciare la curva del drago di ordine nove, potendo così alfine soddisfare la sua curiosità di vedere che tipo di disegno abbia in casa il buon vecchio Knuth. Inoltre è stato così gentile da allegare alla descrizione del suo lavoro alcune foto delle varie schermate, sia per evitarsi la fatica di copiare a mano i listati dei programmi (non avendo la stampante) sia per mostrare i risultati grafici dei suoi programmi anche a chi non dispone dello stesso sistema hardware + software. Pubblico volentieri alcuni di essi perché mi sembra che se lo meritino, e mi sembra giusto assegnare all'autore un abbonamento ad MC per un anno per premiarlo del buon lavoro svolto. Gradirei comunque conoscere l'opinione di altri Texasiani dotati di Logo, o comunque di altri Logofili in genere, su questi programmi per vedere cos'altro si può tirare fuori da questo linguaggio nel

Collaborare a MCintelliGIOCHI

Come vedete, i vostri interventi stanno cominciando ad arrivare e ad essere pubblicati. Vi rinnoviamo pertanto l'invito a collaborare alla rubrica intelliGIOCHI: oltre alle critiche ed ai suggerimenti, sempre bene accetti, ci aspettiamo da voi soprattutto un resoconto delle vostre esperienze nel settore dei giochi intelligenti con il calcolatore. Potete mandarci spunti ed idee ma anche programmi completi; in caso questi fossero troppo lunghi o troppo legati alla macchina per essere pubblicati in queste pagine verranno dirottati alle apposite rubriche di software per le varie macchine. I migliori interventi verranno invece senz'altro pubblicati e discussi nell'ambito di questa rubrica, e compensati in modo proporzionale al loro interesse ed alla loro validità.

MC intelliGIOCHI vuole essere un punto d'incontro e di stimolo per chi ama divertirsi col computer in modo creativo: aspettiamo quindi di conoscere le vostre reazioni ed i vostri interessi in merito.

Le modalità di preparazione ed invio del materiale sono le solite: spiegate brevemente, ma chiaramente il soggetto, indicando non solo cosa avete fatto, ma anche come; listati e/o flowchart sono utili, così come eventuali illustrazioni, foto dello schermo e cose del genere. In caso di programmi complessi sarà bene accludere una descrizione dell'algoritmo seguito e delle routine principali del programma, e magari anche una versione del programma registrata su cassetta o dischetto. Non dimenticatevi di specificare chiaramente su che macchina e che configurazione girano i programmi, e di indicare le vostre generalità ed il vostro recapito non solo sulla busta ma anche sui fogli interni. Per motivi organizzativi non possiamo rispondere a tutte le lettere, e quindi vi preghiamo di non accludere francobolli per la risposta: comunque indicate chiaramente il vostro recapito telefonico, che ci potrà servire per comunicare rapidamente con voi in caso di necessità.

Un esempio di trasformazione ricorsiva: applicando quattro volte la trasformazione di sinistra a se stessa si ottiene la curva di destra

La curva di sinistra è stata ottenuta iterando la trasformazione a triangolo pubblicata nel precedente articolo sulle passeggiate ricorsive (MC 42 pag. 42). Quella di destra mediante la trasformazione ad esagono illustrata nelle due foto in basso in questa pagina.

campo delle passeggiate planari.

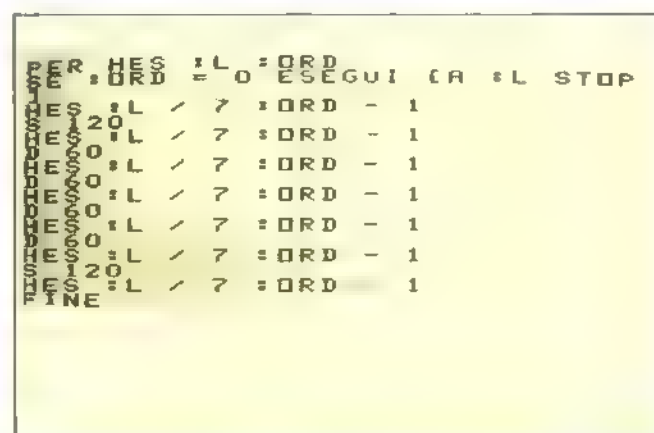
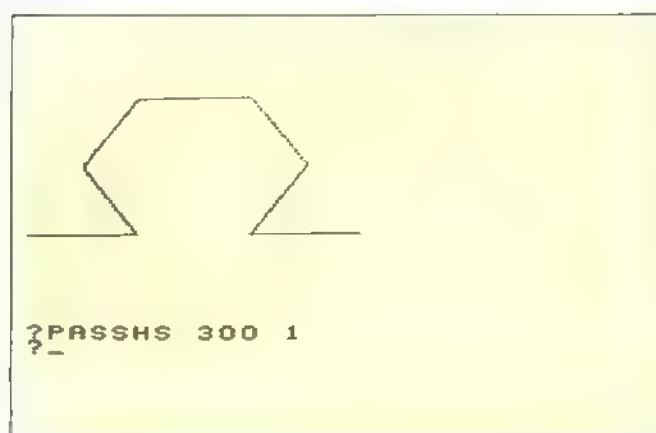
Un secondo intervento interessante è giunto da Mauro Rezzonico del SunQLub Como, si tratta di un paio di routine molto brevi scritte nientemeno che in Lisp per il Sinclair QL: la prima traccia il fiocco di neve esagonale di Koch, la seconda quello quadrato di Mandelbrot. Il Lisp adoperato (QL Lisp Development Kit della Metacomco) è dotato del set completo di istruzioni Turtlegraphics del Logo, ed è quindi in grado di pilotare disegni sullo schermo. Purtroppo questi programmi meritano solo l'onore della citazione, in quanto l'autore non si è profuso molto in spiegazioni; lo invito, se vuole, a riscrivermi accludendo maggiore documentazione e magari qualche foto dei risultati.

Ultimo degli interventi di cui vorrei parlare in questa puntata è quello di Carlo Randone di Chivasso, che ha inviato un corposo programma in Pascal (oltre 280 righe) che implementa tutte le procedure ricorsive discusse nell'articolo. Sviluppato in Hisoft Pascal 4T su di un Amstrad CPC 464, permette il tracciamento delle varie curve in modo guidato da menu. Il programma è ben fatto anche se soffre di una certa scarsezza di portabilità; l'autore non ha però accluso esempi dei risultati ottenibili col programma, e quindi è difficile valutarne le prestazioni ad occhio.

E a proposito del tornare sull'argomento, ho notato che tutti gli interventi ricevuti sull'argomento "passeggiate nel piano" erano incentrati sui percorsi di Peano presentati nella seconda parte, mentre nessuno si è occupato degli argomenti trattati nella prima. Mi meraviglio in particolare del fatto che nessuno abbia sviluppato un programma di Worms: i casi sono due, o l'argomento non è stato ritenuto abbastanza interessante oppure le difficoltà insite nella realizzazione del programma (specialmente per l'elevata risoluzione necessaria in uscita) hanno scoraggiato i volenterosi. Eppure mi sembra che questo giochino non sia affatto banale, rappresentando forse un analogo del più famoso Life nell'ambito dei percorsi planari. Mi piacerebbe sentire qualche opinione in merito, così come ricevere ulteriori resoconti sullo studio delle curve ricorsive.

In particolare c'è ancora molto da dire su quelle del drago: ad esempio nessuno di quelli che hanno scritto si è accorto che si possono unire quattro draghi per la coda in modo da formare un disegno complesso nel quale le quattro curve non si intersecano l'una con l'altra. Chi è in grado di scrivere un programma per generarlo? Fino a quale ordine delle curve si può fare una cosa del genere, sempre ovviamente evitando incroci fra curve diverse? Questi sono solo due fra i quesiti interessanti riguardanti queste curve. Come dicevo prima, può darsi che in futuro si potrà tornare sull'argomento, sempre che la quantità e la qualità del feedback ricevuto lo consentano.

E con ciò termino questa puntata, appuntamento al solito fra trenta giorni per affrontare insieme un argomento ancora diverso.

```
PER HES :L :ORD
SE :ORD = 0 ESEGUI [A :L STOP
HES :L / 7 :ORD - 1
S 120
HES :L / 7 :ORD - 1
D 60
HES :L / 7 :ORD - 1
D 60
HES :L / 7 :ORD - 1
D 60
HES :L / 7 :ORD - 1
S 120
HES :L / 7 :ORD   1
FINE
```

A sinistra la trasformazione ad esagono, a destra il programma ricorsivo che la itera. Tutte le foto mostrate sono state realizzate dall'autore dei programmi, che evidentemente è molto più bravo come programmatore che come fotografo.

«Hal, mi senti? Apri il portello. Ti ordino di aprire il portello!». «Mi dispiace Dave, ma proprio non posso farlo...».
Riconosciuto? È un frammento tratto da «2001 Odissea nello spazio», precisamente il dialogo fra l'astronauta David Bowman, rimasto chiuso al di fuori della Discovery, e Hal 9000, il supercomputer di bordo che per una crisi psicologica sta uccidendo tutto l'equipaggio. Nel romanzo di Clarke Hal è il più avanzato computer disponibile al momento, un modello poco più che sperimentale costruito in due soli esemplari posti uno a terra ed uno a bordo dell'astronave. Progettato in base a principi di programmazione euristica, (HAL sta infatti per Heurystically programmed Algorithmic computer, anche se la sigla nasconde enigmaticamente il nome IBM), esso è in grado di comprendere ed usare il linguaggio naturale degli uomini, nel romanzo si specifica chiaramente che avrebbe potuto sostenere con facilità il Test di Turing.

Il computer racconta...

di Corrado Giustozzi

Generazione di testi, emulazione del discorso, analisi lessicografiche: questo mese mettiamo il computer alle prese col linguaggio degli uomini.

Hal è un esempio, forse il più famoso in quanto tratto dalla fantascienza, di computer che usa il linguaggio umano sia come input che come output, ed è in grado di formare frasi di senso compiuto per esprimere ed illustrare i suoi «concetti». Ma senza invadere il campo della fantascienza, già al giorno d'oggi sono disponibili computer, o meglio programmi, che in qualche modo e per gli scopi più disparati «manipolano» il linguaggio umano: a volte semplicemente imitandolo senza realmente comprenderlo, altre volte addirittura dando la sensazione di una vera comprensione. Alcune volte il linguaggio è l'input al programma, altre solo l'output. Programmi del genere sono oggetto per la maggior parte di studio, ma anche già di applicazioni in vari settori, dai sistemi esperti a complessi gestori di data base. E se da un lato la traduzione automatica, allo studio da oltre vent'anni, sembra ancora ad uno stadio piuttosto rudimentale, è invece perfettamente funzionante un generatore di riassunti, il quale «legge» un brano e lo riassume, afferrandone i concetti principali e scartando quelli secondari, ed oltretutto usando nel testo generato parole od intere frasi che non comparivano in quello originale.

Ma oltre a queste applicazioni «serie» della manipolazione automatica del linguaggio ve ne sono altre più creative che talvolta investono perfino la sfera dell'arte. C'è ad esempio chi studia il modo di adoperare il computer per creare poesie o brevi romanzi: i loro tentativi sono divisi fra due direzioni contrastanti, quella che mira ad ottenere un brano di senso compiuto indistinguibile da un prodotto umano e quella invece del divertissement teso ad ottenere esempi di quella letteratura non-sense cara ai futuristi od ai dadaisti del passato. Appare comunque chiaro come in tutte queste ricerche la molla di fondo sia il gioco, ossia l'innato senso ludico dell'uomo che trova nel calcolatore un meraviglioso strumento di applicazione. D'altra parte molti famosi risultati di queste ricerche erano o sono poi diventati giochi: basta citare il famoso programma Eliza di Joseph Weizenbaum, oramai disponibile per qualunque personal.

Bene, senza andare troppo lontano, questo mese prenderemo in esame alcune delle più semplici tecniche di manipolazione del linguaggio ad uso ricreativo, con le quali fare un po' di sperimentazione nel campo delle parole. Cominciamo quindi col notare che sul linguaggio si può agire a diversi livelli, in dipendenza da cosa si vuole fare. Di volta in volta si può incentrare la propria attenzione sulle singole lettere, sulle parole o su interi periodi: ad ognuno di questi livelli corrispondono strutture e schemi ben precisi e risultati ben diversi. Come esempio pensiamo di voler scrivere un programma che generi un testo casuale: per ognuno dei tre livelli citati in precedenza è possibile applicare una strategia di generazione uguale come concetto generale, ma ben diversa nei risultati. La strategia è molto semplice, e consiste nel prelevare a caso un elemento da un repertorio di elementi possibili e presentarlo poi in uscita. Questa strategia elementare può essere migliorata notevolmente aggiungendole una fase di verifica di correttezza formale: il programma deve conoscere le leggi che regolano i possibili accoppiamenti fra i vari elementi e quindi controllare se quelli via via scelti risultano coerenti tra loro alla luce delle regole. La nostra strategia prevede quindi di due insiemi di conoscenze: l'elenco di tutti gli elementi possibili (da cui attingere) e l'elenco di tutte le regole possibili (per effettuare i controlli); se non è possibile o pratico elencare *tutti* gli elementi o *tutte* le regole ci si può limitare a quelli verosimili o probabili, dipende dai casi.

Come vedete finora ho parlato genericamente di «elementi» e di «regole» senza specificare cosa siano in realtà. Vediamo quindi nella pratica cosa succede quando si applica questa strategia ai tre livelli prima citati. Nel primo caso, ossia agendo a livello di carattere, il programma si limita a generare uno dopo l'altro dei caratteri tratti dall'insieme dei simboli alfabetici, producendo così un insieme disordinato di lettere: le regole con cui effettuare la selezione saranno quelle della scrittura della lingua italiana: non ci possono essere più di tre consonanti assieme, una parola non può finire per consonante, dopo una «q» è obbligatoria una «u» e così via. Più regole si è in grado di specificare e più l'output sarà filtrato, e quindi rassomigliante ad un vero testo italiano. Naturalmente il risultato ottenuto sarà assolutamente privo di significato anche se leggibile. Le parole così formate ben difficilmente apparterranno alla lingua italiana, e sarà in ogni caso impossibile identificare un segmento avente l'apparenza di un discorso. Una interessante variante a questo sistema consiste nel generare i caratteri non in modo perfettamente casuale per poi andare a sfoltire il risultato, ma nel generarli in modo probabilistico o, se volete, pesato, in modo cioè che ogni carattere abbia una propria determinata probabilità di uscire. Questo concetto dà adito a tutta la classe di generatori di testi casuali dalle caratteristiche piuttosto interessanti, di cui parlerò in seguito. Per ora considererò invece generazioni casuali non pesate. Volendo ottenere frasi più verosimili si può passare al livello superiore, lavorando non più sui caratteri, ma sulle parole. E qui le cose incominciano a compli-

carsi, essendo necessario disporre di un repertorio di parole, ossia di un dizionario, da cui il nostro programma possa trarre le parole da usare. Le regole di filtro in questo caso saranno quelle della grammatica e della sintassi della lingua italiana: gli aggettivi e gli articoli si devono riferire ad un nome e devono essere accordati in genere e numero, il predicato verbale deve avere un soggetto ed un complemento e così via. Anche qui, facile a dirsi, ma difficile a farsi: la nostra lingua è piuttosto complessa, ed implementare una soddisfacente ricerca di regole diventa assai arduo non appena si prendono in considerazione gli accordi delle forme verbali, i prefissi ed i suffissi eccetera eccetera. Da questo punto risulta assai più semplice utilizzare l'inglese, dalla grammatica notoriamente molto più semplice della nostra: in questo caso una frase del tipo «articolo-aggettivo - nome - verbo-avverbio-complemento» ha molta probabilità di filare al primo colpo, senza tante preoccupazioni. Comunque balza agli occhi che per rendere efficiente il lavoro non si può procedere generando parole a caso, ma bisogna introdurre degli schemi, ovviamente basati sulla struttura del discorso. Bisogna cioè dividere le parole in base alla loro funzione (nomi, verbi, aggettivi...) e stabilire una «falsariga» di frase, attingendo poi le parole dai gruppi opportuni, infine curando gli accordi grammaticali fra le varie parti.

Da qui al terzo livello prima citato il passo è breve, e consiste nello scegliere come unità di base non più le singole parole, ma interi «moduli» del discorso: intere locuzioni, predicati nominali e verbali, complementi e così via, ognuno dei quali sia una parte complessa del discorso. Questa tecnica è nota come SIMP (Simplified Integrated Modular Prose), ed è stata originariamente sviluppata dalla Honeywell in un breve studio sui «Buzz Phrase Generators» di una quindicina d'anni fa. Nella versione originaria, presentata in figura 1, erano previsti quattro moduli ognuno dei quali composto di dieci diverse frasi. Ogni modulo rappresenta una parte del discorso (introduzione o congiunzione, predicato nominale, predicato verbale e complemento oggetto), e la frase completa si forma scegliendo un elemento dal primo modulo, uno dal secondo, uno dal terzo ed uno dal quarto. Con dieci diverse alternative per ogni modulo si possono generare ben diecimila frasi diverse, tutte assolutamente corrette dal punto di vista sintattico anche se magari carenti da quello semantico. La versione originale del SIMP era costruita per generare prosa tecnica, ma è ovviamente possibile adattare la struttura dei moduli per far assomigliare la prosa generata a ciò che si vuole, da un resoconto di un congresso medico ad un discorso politico. Il risultato è sempre lo stesso: un discorso formalmente corretto ed apparentemente coerente, ma totalmente privo di struttura profonda, tipo «Rispetto a scopi specifici, l'inizio dello sviluppo di un sottosistema critico si complica ulteriormente quando si prenda in considerazione la filosofia dei sistemi e della standardizzazione».

Il vantaggio è che in effetti non è difficile scrivere un programma che generi buona prosa SIMP partendo da un corretto insieme di frasi: raffinatezze interessanti consistono nel disporre ad arte la punteggiatura e nell'evitare di ripetere più volte la stessa locuzione in un discorso. Per la cronaca un programma del genere, ad opera di Valter di Dio, è stato pubblicato nel software Apple sul numero 20 di MC; nel caso particolare l'insieme di moduli è stato scelto per imitare la vuota pomposità di un tipico discorso elettorale.

Tabella SIMP 1:
0 in particolare
1 d'altra parte,
2 tuttavia
3 analogamente
4 come risultato necessario
5 a questo proposito,
6 basandosi su considerazioni di sottosistemi integrali
7 per esempio,
8 quindi
9 rispetto a scopi specifici

Tabella SIMP 2:
0 gran parte del rapporto coordinazione-comunicazione
1: un flusso costante di informazione effettiva
2: la caratterizzazione di criteri specifici
3: l'inizio dello sviluppo di un sottosistema critico
4 un programma di prova completamente integrato
5 la configurazione di base risultante
6 qualsiasi elemento di supporto associato
7: l'incorporazione di vincoli addizionali
8 un principio funzionale indipendente
9: un'interrelazione primaria tra tecnologie del sistema e/o del sottosistema

Tabella SIMP 3:
0 deve utilizzare e legarsi funzionalmente con
1 massimizza le probabilità di successo del progetto e minimizza il costo ed il tempo richiesti per
2: aggiunge specifici limiti d'impiego per
3: fa sì che si consideri con urgenza
4: richiede una notevole quantità di analisi dei sistemi e di studi sugli scambi al fine di ottenere
5: si complica ulteriormente quando si prenda in considerazione
6: presenta dei problemi estremamente interessanti per
7: conduce ad un significativo completamento riguardo
8 limita ulteriormente nel suo impiego
9: riconosce l'importanza di altri sistemi rendendo indispensabile

Tabella SIMP 4:
0: un sofisticato hardware
1: l'anticipato allestimento di una nuova generazione di sistemi
2, un opportuno test di compatibilità per il sottosistema
3: un progetto strutturale basato su tecniche e concetti di ingegneria dei sistemi
4: il limite di qualificazione preliminare
5: l'evoluzione delle specifiche su un dato periodo di tempo
6: la filosofia dei sistemi e della standardizzazione
7: un concetto più generale del rapporto sforzo-premio
8: ogni metodo di configurazione discreta
9: il sistema razionale nel suo complesso

Le quattro tabelle usate dal metodo SIMP di generazione di prosa pseudocasuale. Il concetto è semplice: basta scegliere un elemento a caso da ogni tabella e leggere le frasi nell'ordine. Notiamo infatti che ogni elemento è costruito in modo da saldarsi perfettamente a tutti quelli della tabella precedente e/o seguente. Nel caso presente le frasi sono state scelte in modo da imitare una prosa tecnica, ma è possibile modificare lo scenario a piacere

Anche il SIMP è ovviamente suscettibile di estensioni: ad esempio si può arricchire il numero di frasi in ogni modulo o, meglio, aumentare il numero di moduli. Con più moduli del discorso e frasi ad hoc si possono ottenere risultati piuttosto interessanti: variando «scenario» il programma può ad esempio generare cose come improbabili proverbi («la gallina che si alza nell'armadio non otterrà nulla dal suo computer»), strani anatemi («sia maledetto l'avvocato che sceglie pasticcini a mezzanotte»), e via dicendo. Ciò obbliga naturalmente a rendere più raffinata la struttura della frase: a questo proposito giova sottolineare che non è assolutamente detto che si debba scegliere una sola struttura standard: conviene anzi metterne a punto diverse e fare sì che il programma stesso scelga casualmente per prima cosa il modello di frase e quindi passi a riempirlo con gli opportuno moduli. Il risultato di-

pende comunque dal numero e dalla qualità delle frasi appartenenti ai vari moduli: quattro moduli di dieci frasi l'uno sono proprio il minimo indispensabile per ottenere qualcosa di plausibile.

I programmi scrittori di brevi racconti che citavo prima sono basati su una tecnica tipo SIMP arricchita di controlli sulla trama del racconto: ad esempio occorre memorizzare il sesso e/o il carattere di ogni personaggio per poter decidere quali azioni possa intraprendere, tenere traccia dello stato dei personaggi (la maggior parte delle trame sono di tipo giallo, e quindi i protagonisti sono soggetti a repentine scomparse...) e così via. Pare che qualcuno di questi brevi racconti non sia malvagio: il buon Knuth nella sua monumentale «The Art of Computer Programming» cita il caso in cui un computer fu adoperato per scrivere la base di una sceneggiatura da cui fu tratto un telefilm poi effettivamente realizzato e mandato in onda dalla TV americana.

Ho accennato prima alla generazione casuale di tipo pesato. Questa naturalmente può essere applicata anche alla generazione per parole o per frasi, ma si rivela interessante soprattutto quando viene applicata alla generazione per caratteri. In questo caso si possono scegliere due alternative: dare ad ogni simbolo un peso «assoluto», ad esempio la frequenza caratteristica della lingua in uso, o «relativo» al carattere precedente. Nel primo caso basta disporre di un vettore di probabilità con un valore per ogni carattere generabile, nel secondo serve una matrice cosiddetta «di transizione», che stabilisce la probabilità di uscita di un carattere in base a quello uscito in precedenza. (Per un discorso sulle matrici di transizione vi rimando alla serie di articoli sulla simulazione di Valter di Dio presentati nei numeri scorsi di MC). Una matrice di transizione del genere può essere calcolata in base a semplici analisi statistiche su un particolare testo o su un gran numero di testi di una data lingua. Il testo generato in questo modo è assimilabile ad una catena di Markov, e riflette più da vicino le caratteristiche di tipo statistico della lingua o del testo da cui sono stati estratti i valori di frequenza. Una interessante trattazione di questo tipo di generazione di testi è apparsa nella rubrica «(Ri) Creazioni al Calcolatore» su Le Scienze numero 185 di gennaio 1984, cui rimando senz'altro i più curiosi (ma ne riparleremo nella prossima puntata). La parte difficile in tutto questo discorso sta però proprio nel procurarsi il vettore o la matrice con i valori di probabilità dei vari caratteri. La cosa migliore è un programma che compia analisi statistiche sui testi. Naturalmente chi ha un computer ed un word-processor ha probabilmente diversi testi archiviati, e quindi la cosa più ovvia da fare, piuttosto che copiarsi un libro od un giornale, è adoperare il materiale letterario personale, già bello e pronto. Per chi volesse tentare l'esperimento presento allora un semplice programmino che si incarica di semplificare in qualche modo le operazioni di lettura dei testi scritti con WordStar. Il programma in questione (v. fig. 1) si chiama WStoWORD in quanto è in grado di isolare le parole presenti in un file di WordStar. Più precisamente il programma legge un documento in formato WordStar e lo converte in un file ASCII costituito dalle sole parole presenti nel testo, scritte in lettere maiuscole e private di qualunque carattere non alfabetico. Le parole sono inoltre separate da Carriage Return, così che il file oggetto può essere tranquillamente letto da un programma Basic in modo sequenziale senza alcun problema. Il programma è una versione semplificata del primo di un set che ho preparato per effettuare analisi statistiche sui testi. Scritto in modo da favorire la chiarezza piuttosto che l'efficienza, il suo unico difetto è la lentezza: tipicamente è in grado di convertire ad una media di 14 byte al secondo. Naturalmente è possibile compilarlo per aumentarne la velocità, e allora le cose ritornano ragionevoli. Il prossimo mese riprenderemo il discorso sulla manipolazione dei testi per approfondire la questione sui testi Markoviani e per vedere cos'altro si può fare con i file di parole prodotti da WStoWORD.

MC

```
100 REM *****    WSTOWORD 00.01    *****
110 REM *****       12-06-85       *****
120 REM ***** Corrado  Giustozzi *****
130 '
140 ' Legge un  file di testo in formato  WordStar (.DOC) o ASCII
150 ' e crea un file ASCII sequenziale contenente tutte le parole
160 ' del testo in ordine di  comparsa ed in caratteri maiuscoli,
170 ' scartando tutti i separatori ed i caratteri speciali.
180 '
190 ' Versione semplificata di FREQANOO v. 01.03
200 ' Velocita' di conversione circa 14 byte/secondo
210 '
220 KEY OFF
230 OPTION BASE 1
240 DEFINT A, H, I, S, W
250 DIM WORDLIST$(5000)
260 '
270 ' Richiesta file di input e determinazione file di output
280 CLS
290 INPUT "Filename .DOC : ", R$
300 IF R$="" THEN PRINT "Programma terminato." : END
310 '
320 I = INSTR(R$,".")
330 IF I=0 THEN NOME$ = R$ : EXT$ = "DOC"
    ELSE NOME$ = LEFT$(R$,I-1) : EXT$ = MID$(R$,I+1)
340 F1$ = NOME$ + "." + EXT$
350 F2$ = NOME$ + "." + "WRD"
360 '
370 ' Lettura ed isolamento parole
380 CLS
390 OPEN F1$ FOR INPUT AS #1
400 PRINT "Lettura dal file "; F1$
410 WHILE NOT EOF( 1 )
420     C$ = INPUT$(1, #1)
430     CTR = CTR + 1
440     A = ASC( C$ ) MOD 128
450     ALF = ( A>=65 AND A<=90 ) OR ( A>=97 AND A<=122 )
460     NUM = ( A>=48 AND A<=57 )
470     HYP = ( A=45 )
480     SEP = NOT ( ALF OR NUM OR HYP )
490     IF ALF AND A>90 THEN A = A - 32
500     IF ALF OR NUM THEN WORD$ = WORD$ + CHR$(A) : WRD = -1
510     IF NOT ( SEP AND WRD ) THEN 610
520     IF WORD$<"A" THEN 590
530     ' Isolata una parola
540     CTW = CTW + 1
550     WORDLIST$(CTW) = WORD$
560     LOCATE 7,1
570     PRINT USING "Bytes letti: ##,###"; CTR,
580     PRINT USING "Parole isolate: #,###"; CTW
590     WRD = 0
600     WORD$ = ""
610 WEND
620 CLOSE #1
630 '
640 ' Scrittura file parole
650 LOCATE 10,1
660 PRINT "Scrittura del file "; F2$
670 OPEN F2$ FOR OUTPUT AS #1
680 WRITE #1, F1$, CTW, CTR
690 FOR I = 1 TO CTW
700     PRINT #1, WORDLIST$(I)
710 NEXT I
720 CLOSE #1
730 PRINT
740 PRINT "Termine della conversione."
750 END
```

Figura 1 - Un semplice programma, scritto in BASICA IBM, che legge un file di WorldStar e ne isola le parole. Il suo outpout è un file ASCII contenente tutte le parole trovate nel file originale scritte in caratteri maiuscoli, private di qualsiasi simbolo non alfabetico e separate da Carriage Return. Il file così ottenuto (che per default ha un'estensione .WRD) può essere usato come base per molti tipi di elaborazioni basate sulle parole del testo originale.

La volta scorsa, iniziando il discorso sulla manipolazione del linguaggio, abbiamo mostrato alcuni semplici esempi di come sia possibile scrivere programmi in grado di generare testi pseudocasuali. In particolare abbiamo visto il cosiddetto SIMP o Buzz Phrase Generator, un metodo che nelle sue molte varianti permette di ottenere risultati piuttosto simpatici. Questo mese ripartiamo da quel punto per dirigerci in una direzione piuttosto diversa: lo studio delle caratteristiche statistiche della prosa, con cenni al suo uso nelle applicazioni crittografiche.

Ancora testi al calcolatore

di Corrado Giustozzi

Continua il viaggio nella manipolazione dei testi: questo mese ancora SIMP ed un breve discorso sulla crittografia

rima di entrare nell'argomento principale di questo mese, incentrato sulle proprietà e sui possibili usi di una raccolta di vocaboli, vorrei riprendere brevemente l'argomento SIMP trattato nella scorsa puntata.

Ricordo che la tecnica denominata SIMP (Simplified Integrated Modular Prose) è uno schema che permette di generare prosa sintatticamente corretta, ancorché semanticamente vuota, partendo da un ristretto insieme di «moduli del discorso» costruiti in modo da potersi adattare l'uno all'altro senza disaccordo.

La volta scorsa avevo detto che quanto più è «ricco» questo insieme di moduli tanto più efficace sarà il risultato, sia per la maggiore articolazione delle frasi che per la ristretta probabilità di generare periodi simili tra loro o formati da stessi moduli (naturalmente un buon programma di generazione SIMP può evitare queste imprecisioni, ma con più moduli a sua disposizione anch'esso è in grado di spaziare in maniera ben più ampia fra le varie possibilità).

Mi sembra interessante quindi pubblicare una tabella SIMP piuttosto notevole comparsa su «L'ingegnere italiano» e ripresa anche da «Il Giornale». È stata preparata dai professori Marco Marchi dell'Istituto di biostatistica ed epidemiologia dell'Università di Pisa e Piero Morosini dell'Istituto Superiore di Sanità; essi hanno voluto stigmatizzare un certo tipo di linguaggio politico-programmatico preparando un SIMP in grado di imitarne la struttura. Per far ciò hanno pensato bene di ispirarsi a documenti reali, per cui hanno preso in esame i vari piani di riforma sanitaria presentati negli ultimi anni, ne hanno estratto le frasi ricorrenti e le locuzioni particolarmente sintomatiche e le hanno strutturate in una tabella SIMP a ben sette segmenti. Il risultato è visibile in figura 1: ogni segmento è costituito da dieci elementi diversi, per cui con questa tabella si possono costruire dieci milioni di frasi diverse. I due autori lo hanno presentato in un convegno definendolo, con una certa ironia, «Generatore Automatico di Piani Sanitari». Sorge però un sospetto: non sarà che essi abbiano semplicemente riscoperto per puro caso il reale strumento adoperato dai nostri legislatori per mettere a punto i piani sanitari nazionali?

Ma veniamo all'argomento di questa puntata, che come dicevo in precedenza riguarda le cose che si possono fare avendo a disposizione un archivio di testi già costituito. Accennavo la volta scorsa alla possibilità di compiere analisi statistiche su determinati testi al fine di estrarne le caratteristiche di fondo: elaborazioni di questo tipo, sebbene generalmente piuttosto semplici da effettuare, si rivelano però strumenti fondamentali di lavoro in campi piuttosto specialistici quali ad esempio la crittografia. Questa è una disciplina che si è avvantaggiata enormemente dell'avvento degli elaboratori elettronici, i quali hanno in pochi anni rivoluzionato tecniche e metodi consolidati da secoli. Infatti mentre i «vecchi» metodi si basavano su tecniche di cifratura applicabili a livello di carattere (o parte di carattere, ma in questo caso le operazioni erano assai complesse da svolgere), le tecniche attuali operano facendo ricorso a semplici operazioni binarie su lunghe sequenze di bit che rappresentano i caratteri del testo da cifrare, combinando l'uso degli operatori logici ad operazioni di

L'utenza potenziale	si caratterizza per	il ribaltamento della logica assistenziale preesistente	nel primario interesse della popolazione	sostanziando e vitalizzando	nei tempi brevi, anzi brevissimi	la trasparenza di ogni atto decisionale
Il bisogno emergente	privilegia	il superamento di ogni ostacolo e/o resistenza passiva	senza pregiudicare l'attuale livello delle prestazioni	recuperando ovvero rivalutando	in un'ottica preventiva e non più curativa	la non sanitarizzazione delle risposte
Il quadro normativo	prefigura	un organico collegamento interdisciplinare ad una prassi di lavoro di gruppo	al di sopra di interessi e pressioni di parte	ipotizzando e perseguendo	in un ambito territoriale omogeneo, ai diversi livelli	un indispensabile salto di qualità
La valenza epidemiologica	riconduce a sintesi	la puntuale corrispondenza fra obiettivi e risorse	secondo un modulo di interdipendenza orizzontale	non assumendo mai come implicito	nel rispetto della normativa esistente	una congrua flessibilità delle strutture
Il nuovo soggetto sociale	persegue	la verifica critica degli obiettivi istituzionali e l'individuazione di fini qualificanti	in una visione organica e ricondotta a unità	fattualizzando e concretizzando	nel contesto di un sistema integrato	l'annullamento di ogni ghettizzazione
L'approccio programmatorio	estrinseca	il riorientamento delle linee di tendenza in atto	con criteri non dirigistici	non sottacendo ma anzi puntualizzando	quale sua premessa indispensabile e condizionante	il coinvolgimento attivo di operatori e utenti
L'assetto politico-istituzionale	si propone	l'accorpamento delle funzioni ed il decentramento decisionale	al di là delle contraddizioni e difficoltà iniziali	potenziando ed incrementando	nella misura in cui ciò sia fattibile	l'appianamento di discrepanze e discresie esistenti
Il criterio metodologico	presuppone	la ricognizione del bisogno emergente e della domanda non soddisfatta	in maniera articolata e non totalizzante	non dando certo per scontato	con le dovute ed imprescindibili sottolineature	la ridefinizione di una nuova figura professionale
Il modello di sviluppo	porta avanti	la riconversione ed articolazione periferica dei servizi	attraverso i meccanismi della partecipazione	evidenziando ed esplicitando	in termini di efficacia e di efficienza	l'adozione di una metodologia differenziata
Il metodo partecipativo	auspica	un corretto rapporto fra struttura e sovrastrutture	senza precostituzione delle risposte	attivando ed implementando	a monte e a valle della situazione contingente	la demedicalizzazione del linguaggio

Figura 1 - Il Generatore Automatico di Piani Sanitari, un notevole SIMP a sette segmenti.

scrambling secondo schemi pseudocasuali.

I moderni metodi crittografici sono pertanto assai diversi da quelli, pur sofisticatissimi, che venivano usati nel XVI secolo dagli ambasciatori della Repubblica di Venezia o dai Messi Pontifici; ed anche da quelli in uso durante l'ultima guerra, i quali altro non erano che riedizioni dei precedenti, ossia in definitiva sempre quelli escogitati dall'Alberti, dal Porta o dal Vigenére, veri luminari della materia.

Non ci addentreremo oltre in questo discorso che ci porterebbe troppo lontano: se riceveremo i vostri consensi vedremo di dedicare una puntata intera alla crittografia ed ai metodi crittografici. Per quanto ci riguarda adesso basta solo dire che uno dei più antichi metodi crittografici documentati, piuttosto semplice in verità, può essere rapidamente risolto mediante una elementare analisi delle proprietà statistiche del testo cifrato. Sto parlando del cosiddetto cifrario «a trasposizione semplice» detto anche «di Cesare» in quanto, come ci dice Svetonio, Giulio Cesare ne faceva abitualmente uso nella corrispondenza con i suoi luogotenenti. Esso consiste semplicemente nel sostituire ad ogni lettera del testo da cifrare quella che la precede (o la segue) nell'alfabeto di un certo numero fisso di posizioni. Cesare in particolare lo usava con uno spostamento di quattro lettere in avanti, per cui ogni «A» del testo chiaro diventava una «E», ogni «B» diventava «F» e così via ciclicamente fino alla «Z» che diventava «D». Sembrerà strano, ma questo semplice sistema è rimasto impenetrabile fino al Medio Evo, quando ci si accorse di alcune elementari proprietà statistiche della lingua scritta e parlata ed in particolare si comprese che certe lettere comparivano più spesso di altre anche in testi di lunghezza modesta. Questa pur semplice osservazione è stata per più di otto secoli al centro di tutti i metodi di decrittazione. Grazie ad essa la soluzione di un cifrario di Cesare diventa un gioco da ragazzi: basta conoscere qual è la lettera più frequente nella lingua in cui è scritto il testo e quindi contare i simboli del testo cifrato. Con grande probabilità il simbolo più ricorrente rappresenterà la lettera più frequente, ed a questo punto il gioco è fatto: per via della rigida corrispondenza fra alfabeto chiaro ed alfabeto cifrante, infatti, una volta identificata una lettera sono automaticamente note tutte le altre.

Poco più difficile da risolvere è il cifrario a «sostituzione semplice» in cui la relazione fra alfabeto chiaro e alfabeto cifrante è leggermente meno rigida; in questo caso esiste una tabella di conversione che associa ogni simbolo chiaro al corrispondente cifrato, senza che fra i due ci sia una stretta connessione posizionale. A questa categoria appartiene ad esempio il noto «aneddoto cifrato» della Settimana Enigmistica, che non è altro che un breve testo in cui le lettere sono state sostituite da numeri mediante un procedimento analogo alla trasposizione semplice.

L'applicazione dell'analisi statistica continua ancora a valere, ma ora occorre conoscere le frequenze percentuali di tutte le lettere dell'alfabeto della lingua in uso. È infatti necessario poter identificare con relativa certezza almeno tre o quattro simboli chiave nel testo cifrato, noti i quali si può procedere nella decrittazione con considerazioni di carattere grammaticale o, ancora, statistico ma di tipo

Testo GIOCHI47.DOC di 9516 caratteri:

		Num.	Fr. %
1)	A	931	9.78
2)	B	95	1.00
3)	C	440	4.62
4)	D	373	3.92
5)	E	1,193	12.54
6)	F	109	1.15
7)	G	143	1.50
8)	H	93	0.98
9)	I	1,093	11.49
10)	J	0	0.00
11)	K	2	0.02
12)	L	533	5.60
13)	M	235	2.47
14)	N	589	6.19
15)	O	830	8.72
16)	P	295	3.10
17)	Q	62	0.65
18)	R	650	6.83
19)	S	564	5.93
20)	T	739	7.77
21)	U	322	3.38
22)	V	130	1.37
23)	W	7	0.07
24)	X	1	0.01
25)	Y	3	0.03
26)	Z	84	0.88

Figure 2, 3 - Una tabella delle frequenze caratteristiche della lingua Italiana del 1928 a confronto con le frequenze di questo articolo: ci sono differenze significative?

più sofisticato (su coppie di lettere). Vediamo quindi l'importanza che per gli antichi crittografi aveva conoscere determinate proprietà statistiche delle diverse lingue: solo in base a queste informazioni poteva iniziare il processo di decrittazione di un testo cifrato.

Le tabelle delle frequenze percentuali delle lettere nelle varie lingue costituivano per essi un prezioso strumento di lavoro, e non c'è da meravigliarsi che i (pochi) testi di crittografia ne riportassero diverse. Se pensiamo alla difficoltà di ottenerle per via esclusivamente manuale saremo in grado di apprezzare lo sforzo compiuto ad esempio dal Zanotti che nel suo manuale «Crittografia — le scritture segrete» del 1928 pubblicava le frequenze delle lettere singole e delle coppie, misurate su testi campione di ben 10.000 lettere nelle lingue Italiana, Francese, Inglese, Spagnola, Tedesca e Serbo-croata. A titolo di curiosità riporto in figura 2 quella relativa all'Italiano, tratta dalla ristampa anastatica dell'originale Hoepli (Cisalpino-Goliardica, 1976).

Un lavoro del genere, che cinquant'anni fa richiedeva allo studioso settimane di faticoso lavoro, diventa una questione di pochi minuti per un moderno possessore di personal. Scrivere un programma che legga un testo e calcoli le frequenze percentuali delle lettere è piuttosto semplice; per procurarsi il testo, come dicevo la volta scorsa, basta cercare tra i file di lavoro del proprio word processor: chi di noi non ha infatti un po' di testi memorizzati? Il sottoscritto magari farà eccezione, avendo prodotto crica 650 Kbyte di testi dal solo gennaio di quest'anno (!), ma credo che chiunque usi il word processor poco più che casualmente sia in grado di disporre di un buon volume di testi. Disponendo di una buona sorgente si possono eseguire elaborazioni piuttosto significative: oltre a scoprire le frequenze delle lettere singole si possono andare a vedere quelle delle coppie e delle terne, si può studiare la distribuzione della lunghezza delle parole, si possono analizzare le distribuzioni delle parole stesse o delle sillabe che le compongono. I parametri così ottenuti magari non ci serviranno a decrittare messaggi segreti, ma potranno darci un'idea quantitativa della struttura dei nostri testi o di quelli degli altri. A titolo di esempio pubblico in figura 3 la tabella delle frequenze assolute e percentuali delle lettere che compaiono in questo articolo: lascio a voi l'onere di decidere se la mia prosa differisce da quella del 1928 in misura significativa.

E concludo con una breve appendice alla scorsa puntata, relativa al semplice programmino WstoWord che, se vi ricordate, serve a tirare fuori le parole da un file WordStar scrivendole in maiuscolo su di un file ASCII sequenziale. Come già dicevo in chiusura di puntata il suo unico handicap è l'estrema lentezza di esecuzione, circa 14 byte al secondo. Credo di fare una cosa interessante pubblicandone una versione in C, decisamente molto più veloce. Lo potete vedere in figura 4: io ho usato il compilatore C Lattice/Microsoft su di un PC IBM, ma il programma è così semplice da non creare problemi di alcun genere. Ho scelto esplicitamente di scriverlo come filtro DOS, quindi per leggere e scrivere su disco è necessario sfruttare la redirezione dell'I/O. Nel mio caso il codice oggetto generato è di soli 4 Kb, e la velocità di conversione è di oltre 1400 byte al secondo, ossia più di cento volte maggiore dell'interprete BASICA.

Bene, anche per questa volta chiudiamo qui. Abbiamo discusso di cose, credo, piuttosto interessanti, anche se ancora non abbiamo detto tutto sulle cose che si possono fare con i file prodotti da WStoWORD. Questo sarà in effetti l'argomento della prossima puntata. In essa vedremo come creare un... dizionario con le parole estratte dai testi, e come potremo utilizzarlo per giocare. Vi rinnovo l'invito a scrivere: se avete qualche spunto che ritenete valido riguardo agli argomenti trattati in questo periodo saremo lieti di conoscerlo. Nel frattempo, arrivederci al prossimo mese.

MC

```
/*****    WStoWORD 01.01    *****
 *****      16-11-85        *****
 ***** Corrado  Giustozzi  *****/

#include <stdio.h>
#include <ctype.h>

#define  TRUE      1
#define  FALSE     0
#define  RETURN    '\n'
#define  isseparator(c) ( ( isalnum(c) || ( c == '-' ) ) ? FALSE : TRUE )

/*

Legge un file di testo in formato WordStar o ASCII e crea un
file ASCII sequenziale contenente  tutte le parole del testo
in ordine di comparsa  ed in caratteri  maiuscoli, scartando
le cifre, i caratteri non alfabetici e quelli speciali.

Cfr. MC IntelliGIOCHI, MC n. 46 (novembre 1985) e n. 47 (dicembre 1985)

*/

main()

{    /* main */

int      c, In_Word;

while ( ( c = getchar() ) != EOF ) {
      c = toupper( toascii( c ) );
      if ( isalpha( c ) ) {
           In_Word = TRUE;
           putchar( c );
      }
      else if ( isseparator( c ) && In_Word ) {
           In_Word = FALSE;
           putchar( RETURN );
      }
}

}    /* main */
```

Figura 4 - Una nuova versione del WStoWORD, questa volta scritta in C: più veloce di oltre cento volte rispetto a quella in BASICA pubblicata la volta scorsa.